



MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025



Table of contents

1	LPCXpresso55S36	3
1.1	Overview	3
1.2	Getting Started with MCUXpresso SDK Package	3
1.2.1	Getting Started with Package	4
1.3	Getting Started with MCUXpresso SDK GitHub	46
1.3.1	Getting Started with MCUXpresso SDK Repository	46
1.4	Release Notes	59
1.4.1	MCUXpresso SDK Release Notes	59
1.5	ChangeLog	64
1.5.1	MCUXpresso SDK Changelog	64
1.6	Driver API Reference Manual	139
1.7	Middleware Documentation	139
1.7.1	Multicore	139
1.7.2	Audio Voice components	139
1.7.3	Maestro Audio Framework for MCU	139
1.7.4	FreeMASTER	139
1.7.5	FreeRTOS	139
1.7.6	File systemFatfs	139
2	LPC55S36	141
2.1	ANACTRL: Analog Control Driver	141
2.2	AOI: Crossbar AND/OR/INVERT Driver	145
2.3	CACHE: CACHE Memory Controller	149
2.4	CACHE: LPCAC CACHE Memory Controller	153
2.5	CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing	153
2.6	casper_driver	153
2.7	casper_driver_pkha	157
2.8	CDOG	160
2.9	Clock Driver	163
2.10	CMP: Analog Comparator Driver	201
2.11	CRC: Cyclic Redundancy Check Driver	205
2.12	CTIMER: Standard counter/timers	208
2.13	DAC: Digital-to-Analog Converter Driver	218
2.14	DMA: Direct Memory Access Controller Driver	223
2.15	DMIC: Digital Microphone	240
2.16	DMIC DMA Driver	240
2.17	DMIC Driver	242
2.18	Efuse_driver	252
2.19	ENC: Quadrature Encoder/Decoder	253
2.20	Flash_driver	263
2.21	Flash_ffr_driver	275
2.22	FLEXCOMM: FLEXCOMM Driver	287
2.23	FLEXCOMM Driver	287
2.24	FLEXSPI: Flexible Serial Peripheral Interface Driver	288
2.25	FLEXSPI DMA Driver	305

2.26	Flexspi_nor_flash_driver	307
2.27	FREQME: Frequency Measurement	325
2.28	GINT: Group GPIO Input Interrupt Driver	325
2.29	Hashcrypt: The Cryptographic Accelerator	328
2.30	Hashcrypt Background HASH	328
2.31	Hashcrypt common functions	328
2.32	Hashcrypt AES	331
2.33	Hashcrypt HASH	335
2.34	HSCMP: High Speed Analog Comparator Driver	337
2.35	I2C: Inter-Integrated Circuit Driver	342
2.36	I2C DMA Driver	342
2.37	I2C Driver	343
2.38	I2C Master Driver	347
2.39	I2C Slave Driver	356
2.40	I2S: I2S Driver	366
2.41	I2S DMA Driver	366
2.42	I2S Driver	369
2.43	I3C: I3C Driver	378
2.44	I3C Common Driver	380
2.45	I3C Master DMA Driver	383
2.46	I3C Master Driver	386
2.47	I3C Slave DMA Driver	412
2.48	I3C Slave Driver	413
2.49	INPUTMUX: Input Multiplexing Driver	426
2.50	Inline Prince Encryption Decryption	506
2.51	IRTC: IRTC Driver	510
2.52	Intrusion and Tamper Response Controller	519
2.53	ITRC	519
2.54	Common Driver	522
2.55	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	535
2.56	Lpc_freqme	554
2.57	GPIO: General Purpose I/O	559
2.58	GPIO: General Purpose I/O	563
2.59	IOCON: I/O pin configuration	563
2.60	MCAN: Controller Area Network Driver	564
2.61	Memory_interface	588
2.62	MRT: Multi-Rate Timer	596
2.63	Nboot	600
2.64	Nbot_hal	607
2.65	OPAMP: Operational Amplifier	612
2.66	OSTIMER: OS Event Timer Driver	615
2.67	PINT: Pin Interrupt and Pattern Match Driver	619
2.68	PLU: Programmable Logic Unit	628
2.69	Power Driver	634
2.70	POWERQUAD: PowerQuad hardware accelerator	654
2.71	PRINCE: PRINCE bus crypto engine	684
2.72	PUF: Physical Unclonable Function	691
2.73	Puf_v3_driver	691
2.74	PWM: Pulse Width Modulator	697
2.75	Reset Driver	722
2.76	RNG: Random Number Generator	728
2.77	RTC: Real Time Clock	729
2.78	Runbootloader	733
2.79	Sbloader	734
2.80	SCTimer: SCTimer/PWM (SCT)	745
2.81	SPI: Serial Peripheral Interface Driver	762
2.82	SPI DMA Driver	762
2.83	SPI Driver	765

2.84	SYSCTL: I2S bridging and signal sharing Configuration	774
2.85	USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver	776
2.86	USART DMA Driver	776
2.87	USART Driver	779
2.88	UTICK: MicroTick Timer Driver	796
2.89	VREF: Voltage Reference Driver	797
2.90	WWDG: Windowed Watchdog Timer Driver	799
3	Middleware	805
3.1	Motor Control	805
3.1.1	FreeMASTER	805
3.2	Multimedia	842
3.2.1	Audio Voice	842
4	RTOS	897
4.1	FreeRTOS	897
4.1.1	FreeRTOS kernel	897
4.1.2	FreeRTOS drivers	897
4.1.3	backoffalgorithm	897
4.1.4	corehttp	897
4.1.5	corejson	897
4.1.6	coremqtt	898
4.1.7	corepkcs11	898
4.1.8	freertos-plus-tcp	898

This documentation contains information specific to the lpcxpresso55s36 board.

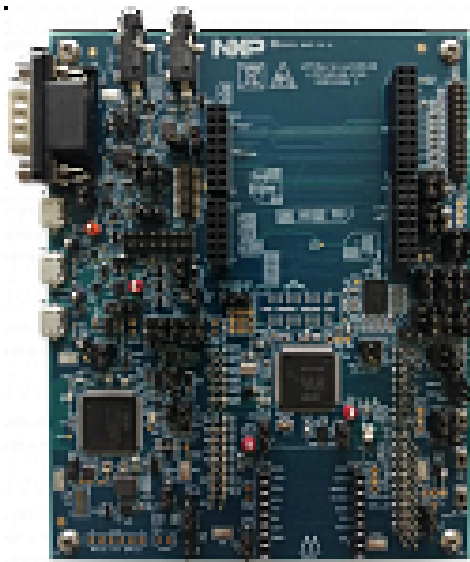
Chapter 1

LPCXpresso55S36

1.1 Overview

The LPCXpresso55S36 development board provides the ideal platform for evaluation of and development with the LPC553x/S3x MCU based on the Arm Cortex-M33 architecture. The board includes a high-performance onboard debug probe and accelerometer, with several options for adding off-the-shelf add-on boards for networking, sensors, displays, and other interfaces.

The LPCXpresso55S36 is fully supported by the MCUXpresso suite of tools, which provides device drivers, middleware and examples to allow rapid development, plus configuration tools and an optional free IDE. MCUXpresso software is compatible with the open source MCU operating system FreeRTOS, tools from popular tool vendors such as Arm and IAR, and the LPCXpresso55S36 may also be used with the popular debug probes available from SEGGER and P&E Micro.



MCU device and part on board is shown below:

- Device: LPC55S36
- PartNumber: LPC55S36JBD100

1.2 Getting Started with MCUXpresso SDK Package

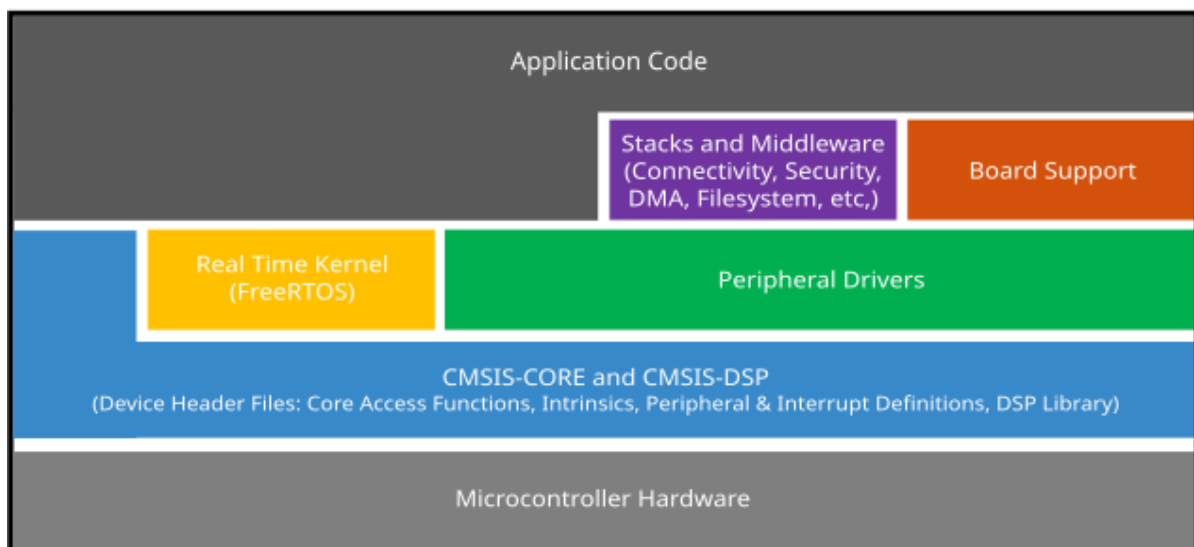
1.2.1 Getting Started with Package

Overview

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

For supported toolchain versions, see *MCUXpresso SDK Release Notes* (document MCUXSDKRN).

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):

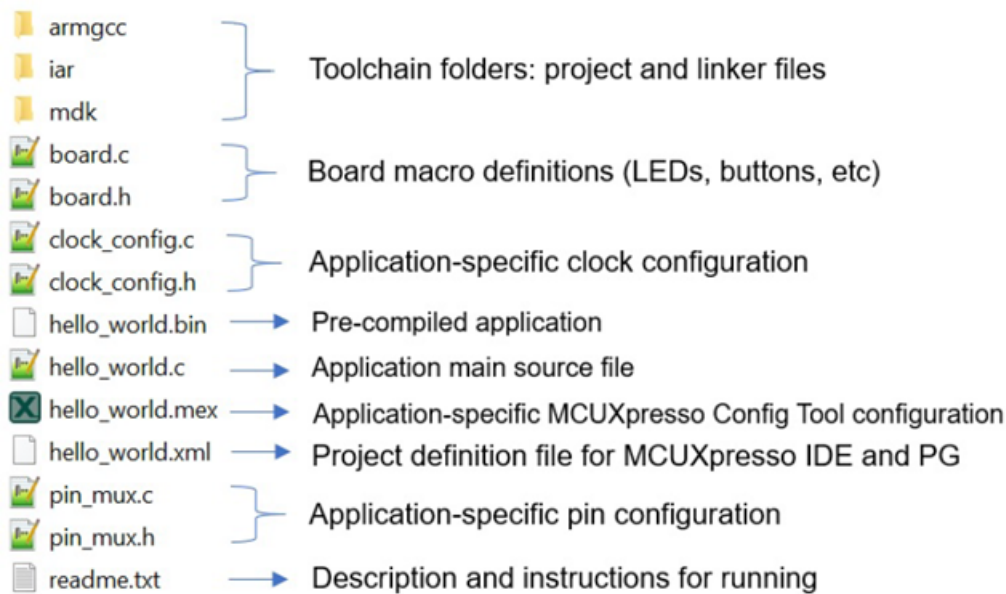
- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- `driver_examples`: 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).
- `emwin_examples`: Applications that use the emWin GUI widgets.
- `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.

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>/cmsis_drivers`: All the CMSIS drivers for your specific MCU
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code, including vector table definitions
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications

- `devices/<devices_name>/project`: Project template used in CMSIS PACK new project creation

For examples containing middleware/stacks or an RTOS, there are references to the appropriate source code. Middleware source files are located in the `middleware` folder and 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: Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

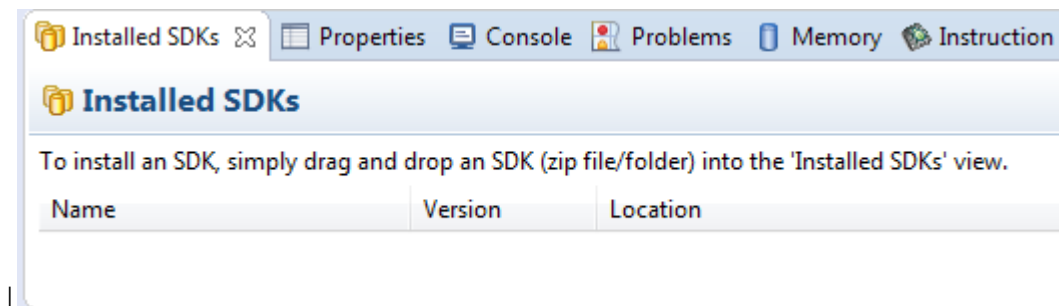
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The `hello_world` demo application targeted for the LPCXpresso55S36 hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso 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.

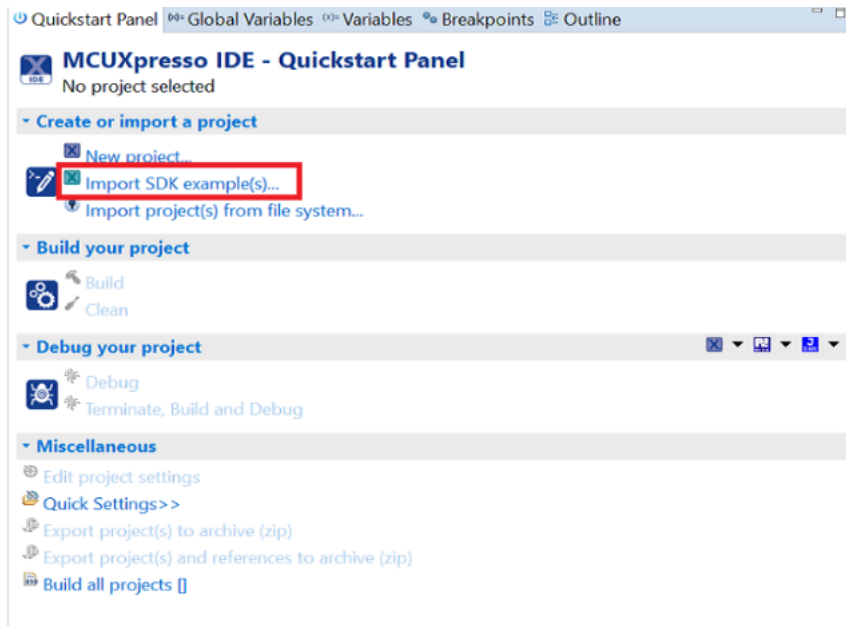
Parent topic: [Run a demo using MCUXpresso IDE](#)

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

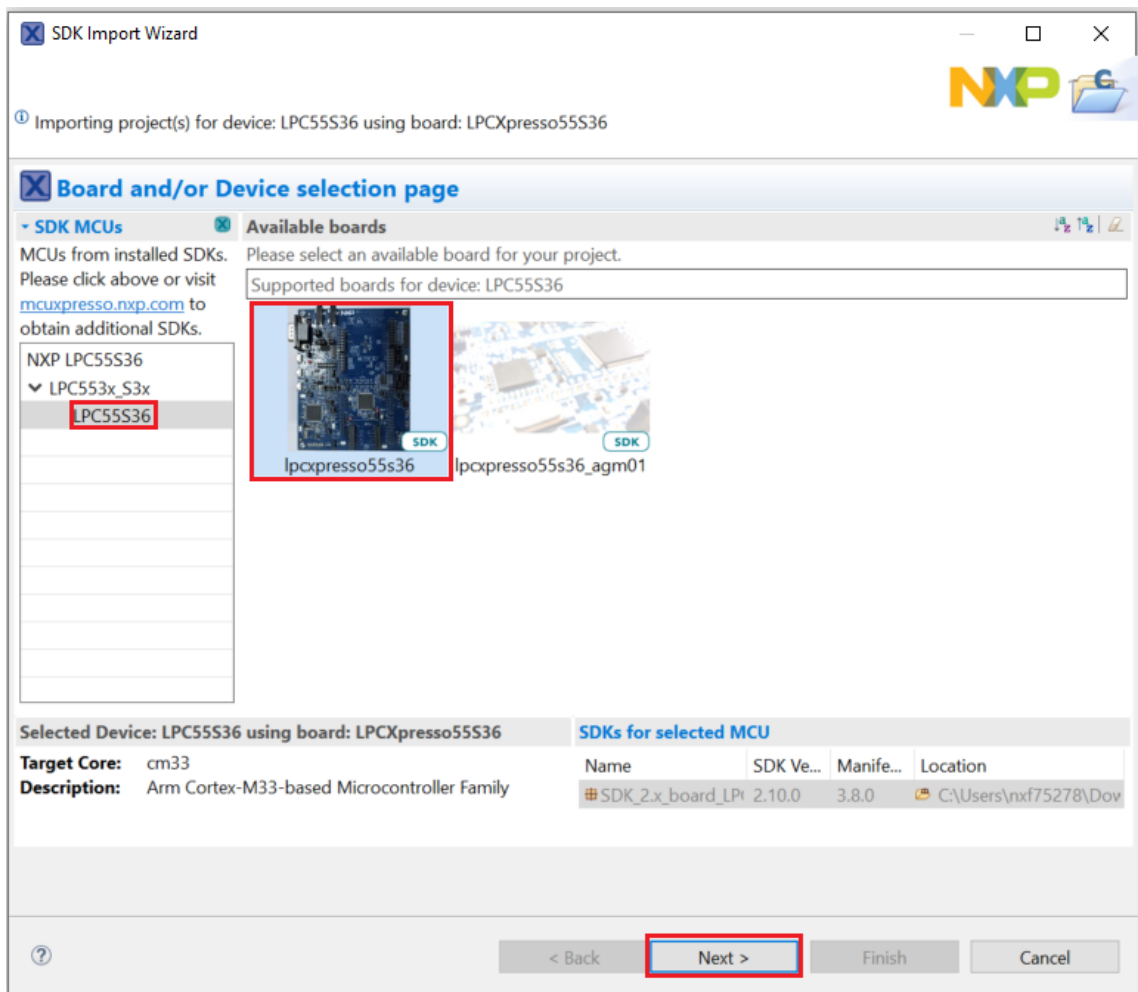
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



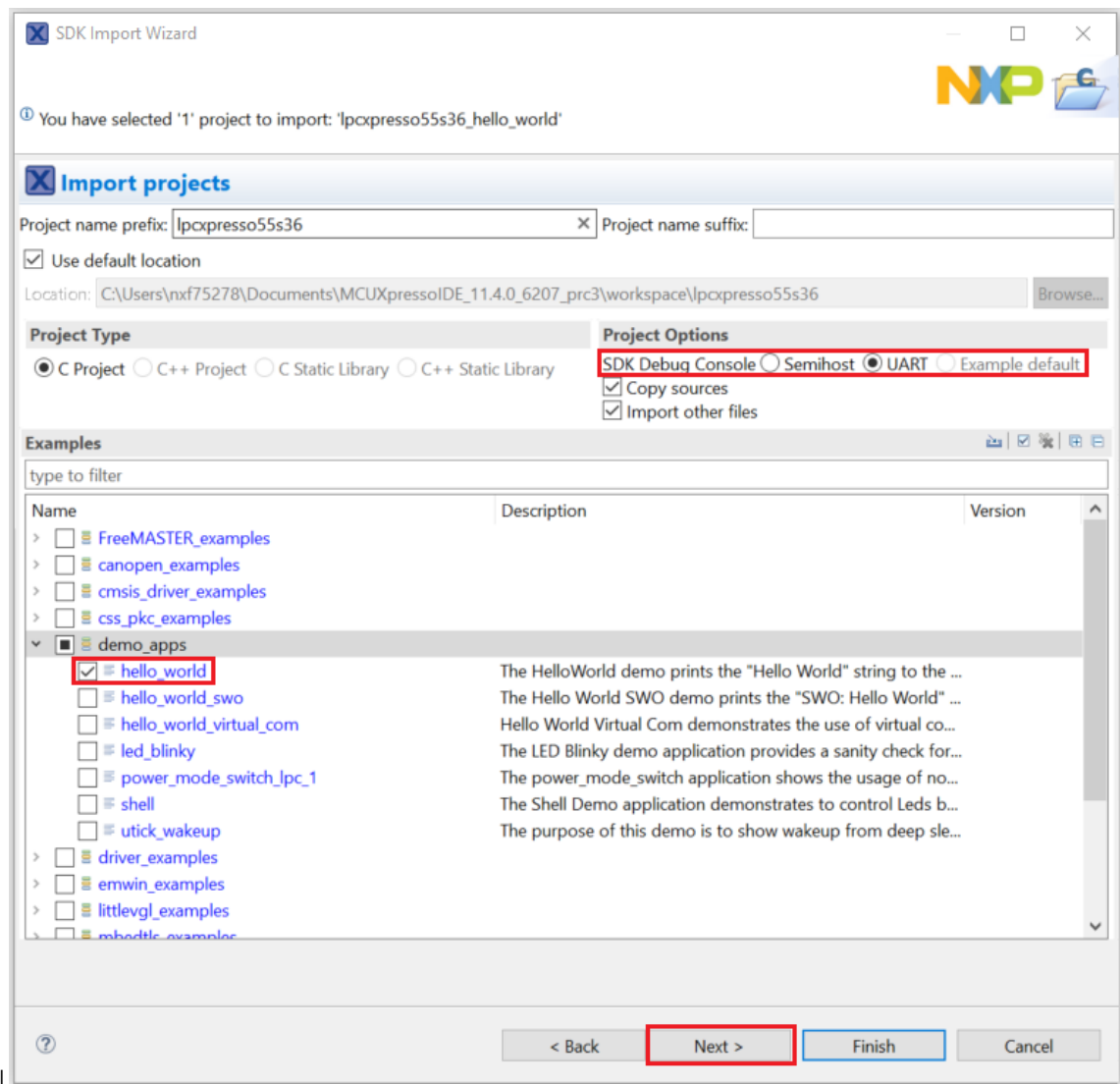
2. On the **Quickstart Panel**, click **Import SDK example(s)...**



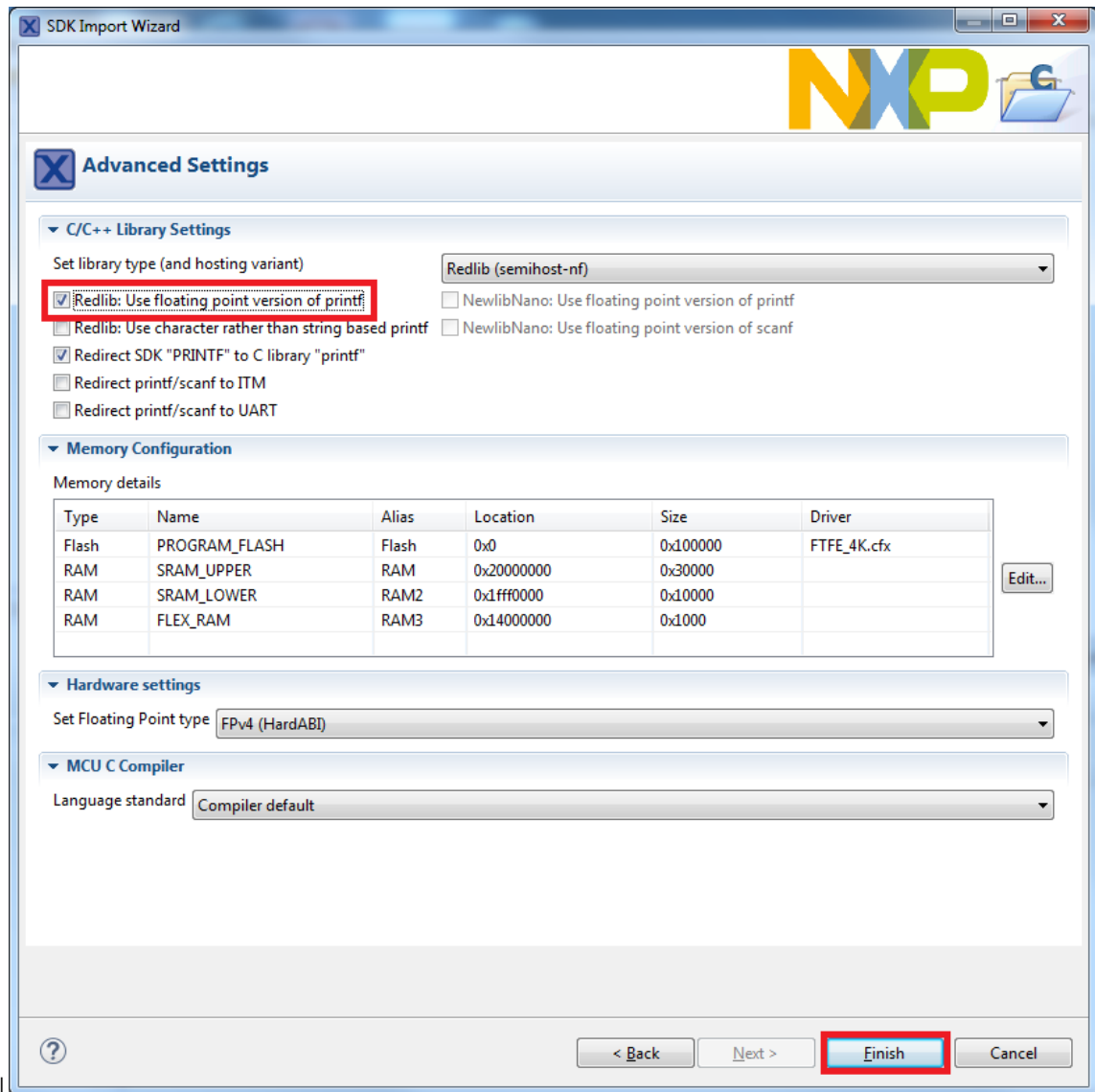
3. In the window that appears, expand the **LPC55xx** folder and select **LPCX55S36**. Then, select **lpcxpresso55s36** and click **Next**.



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**.



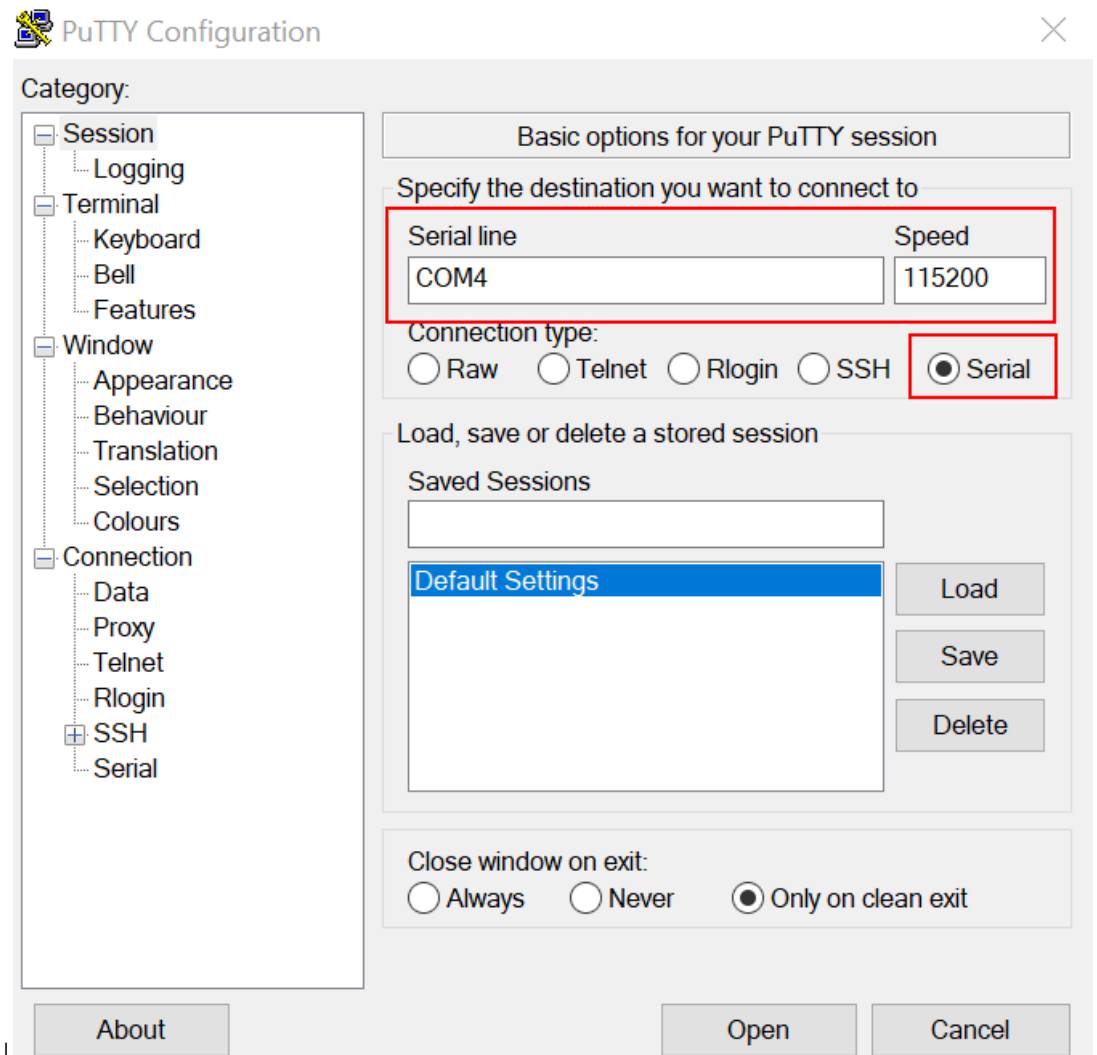
Parent topic: [Run a demo using MCUXpresso IDE](#)

Run an example application For more information on debug probe support in the MCUXpresso IDE, see community.nxp.com.

To download and run the application, perform the following steps:

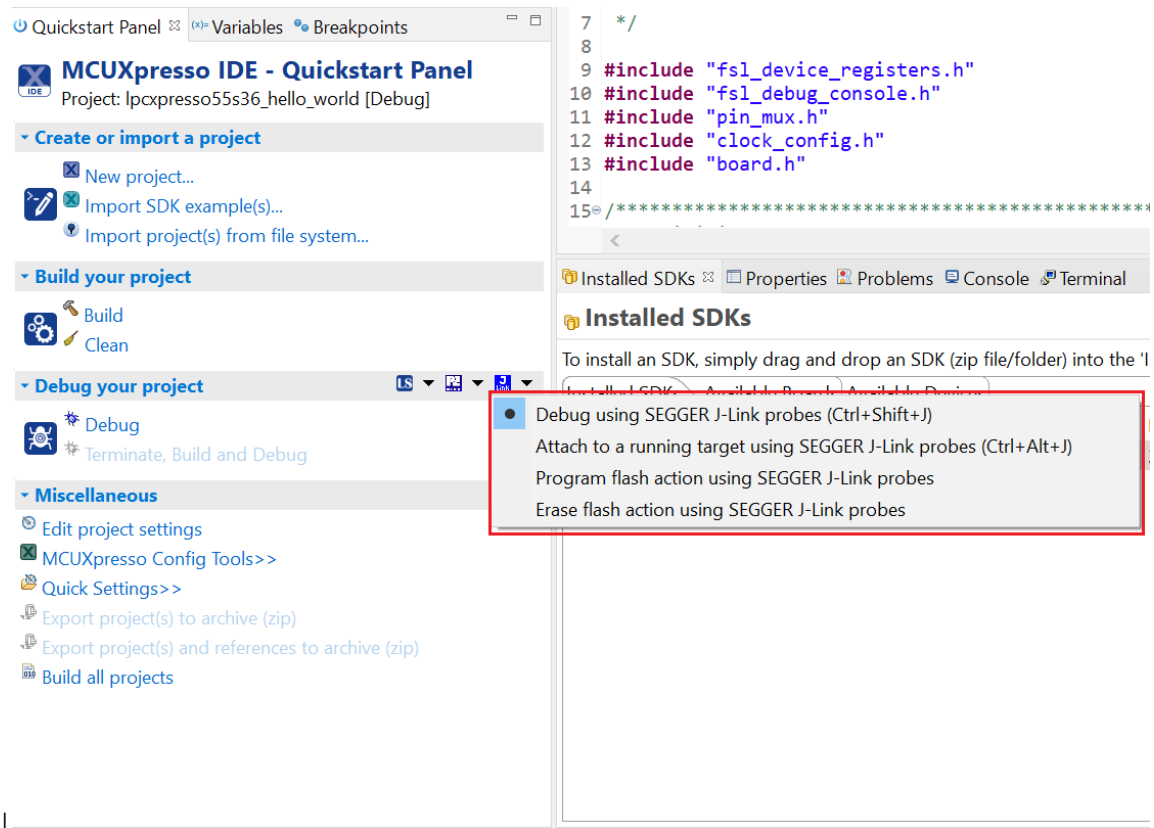
1. See the table in [Default debug interfaces](#) to determine the debug interface that comes loaded on your specific hardware platform. For LPCXpresso boards, install the DFU jumper for the debug probe, then connect the debug probe USB connector.
 - For boards with a P&E Micro interface, see [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.
2. Connect the development platform to your PC via a 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 board.h file)
2. No parity
3. 8 data bits

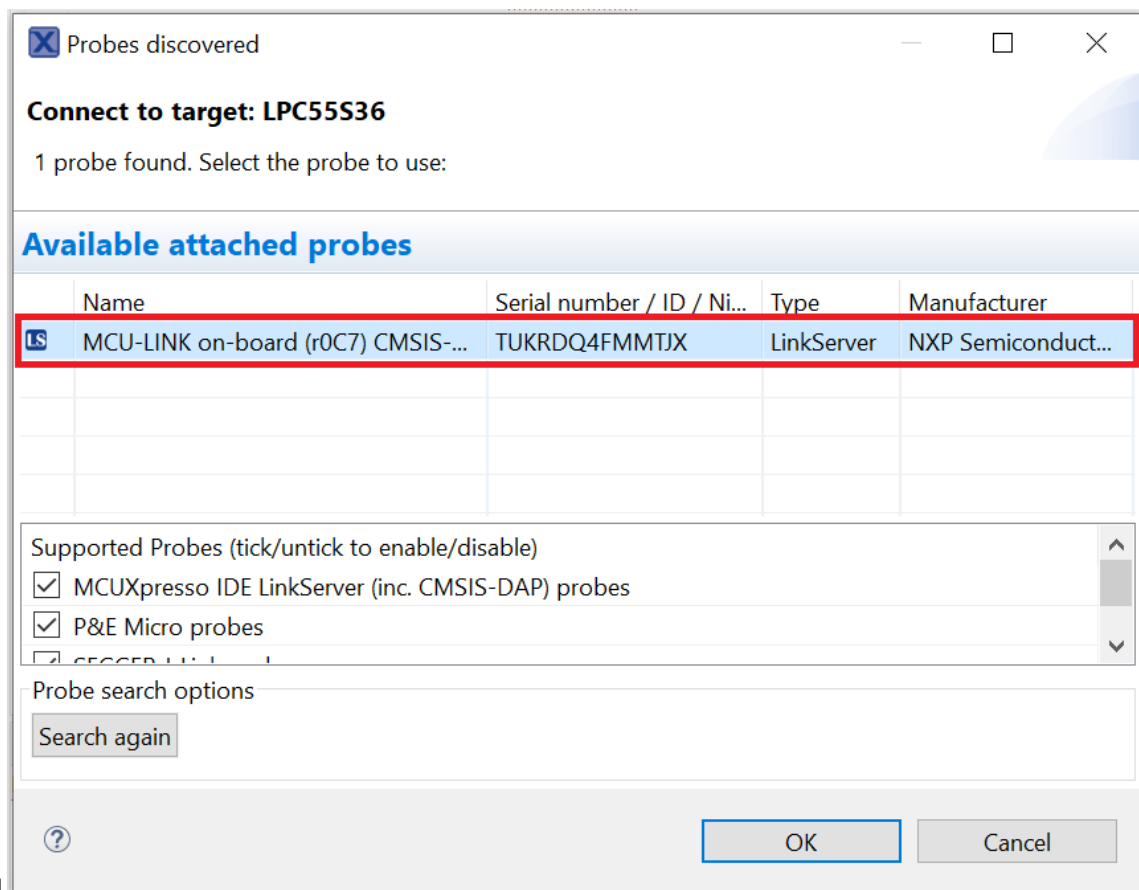


4. 1 stop bit

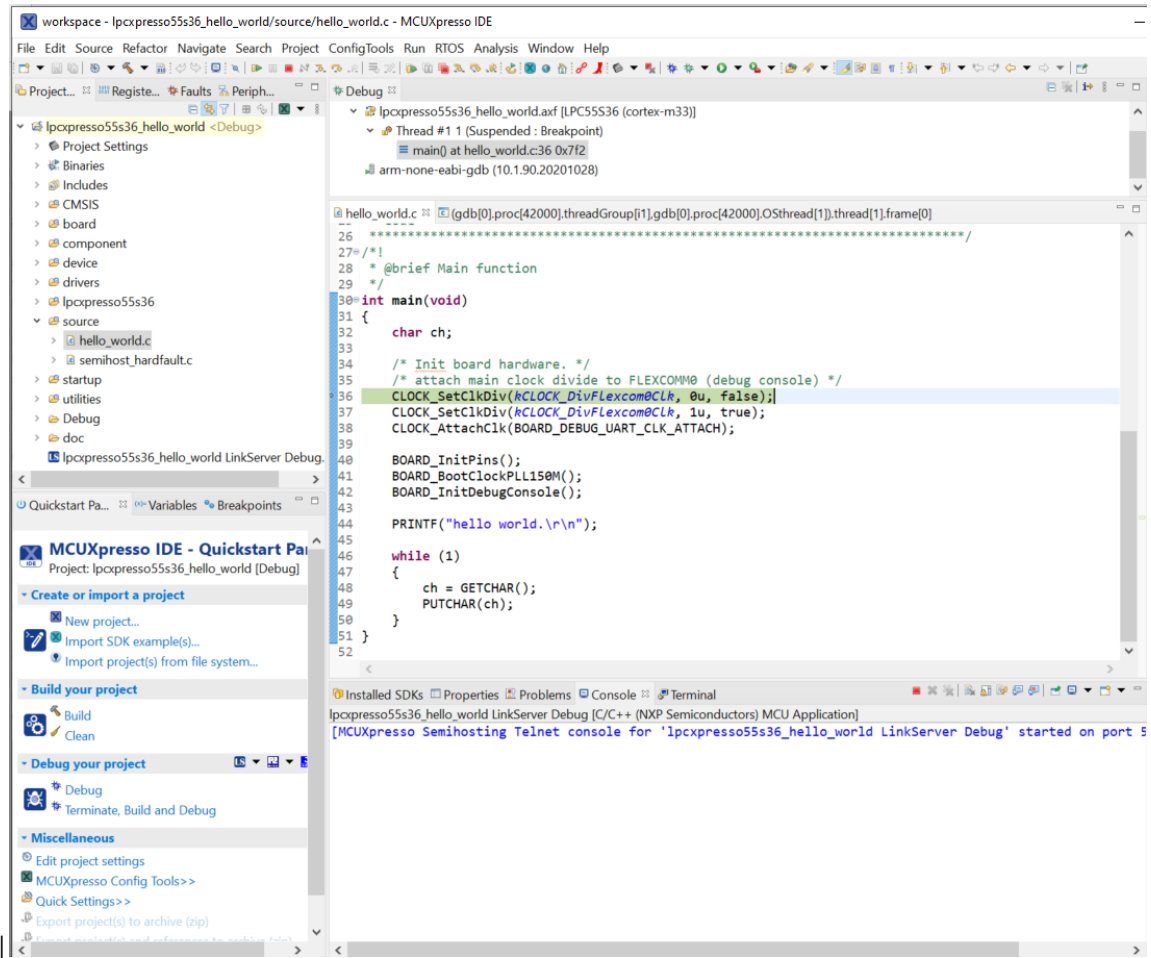
4. On the **Quickstart Panel**, click on **Debug** lpcxpresso55s36_hello_world [Debug] to launch the debug session.



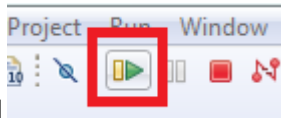
5. The first time you debug a project, the **Debug Emulator Selection** dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. (For any future debug sessions, the stored probe selection is automatically 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**.



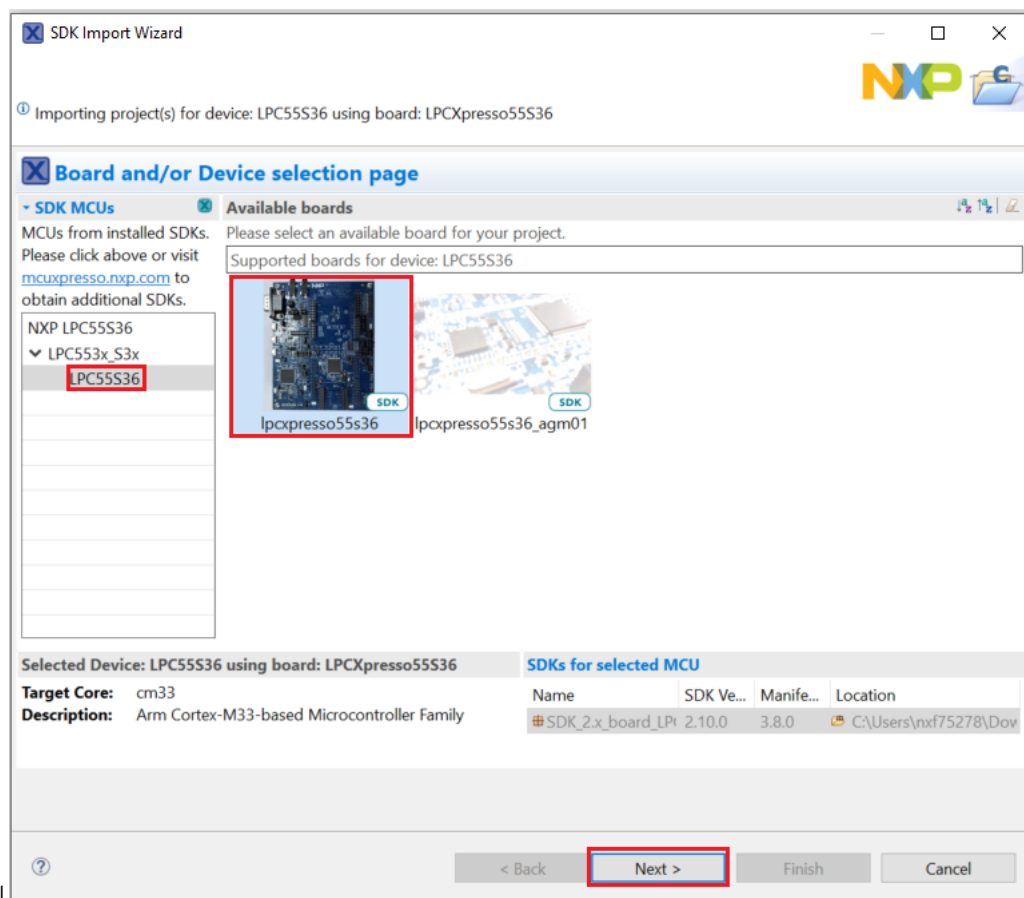
The hello_world application is now running and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



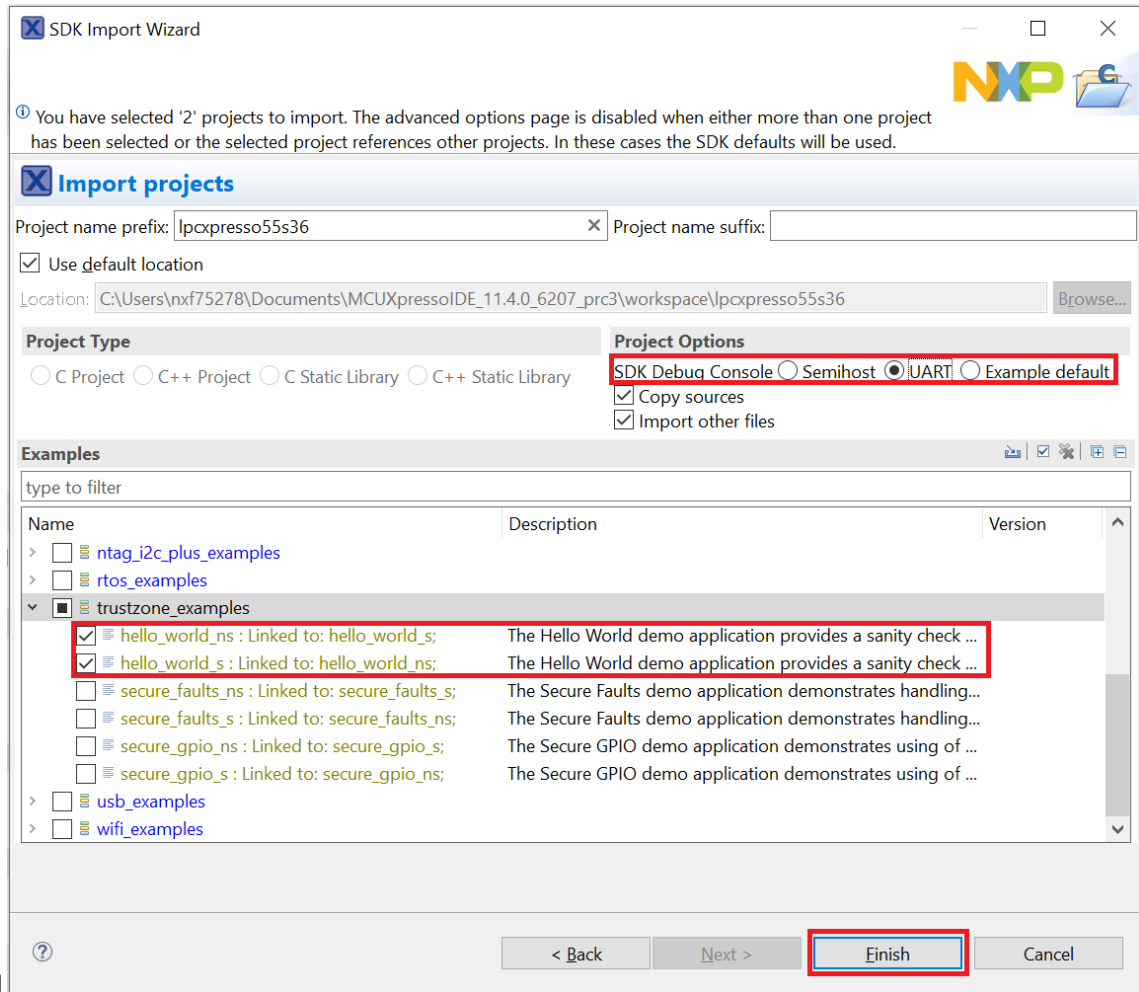
Parent topic: [Run a demo using MCUXpresso IDE](#)

Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The trustzone version of the hello_world example application targeted for the LPCXpresso55S36 hardware platform is used as an example, though these steps can be applied to any TrustZone example application in the MCUXpresso SDK.

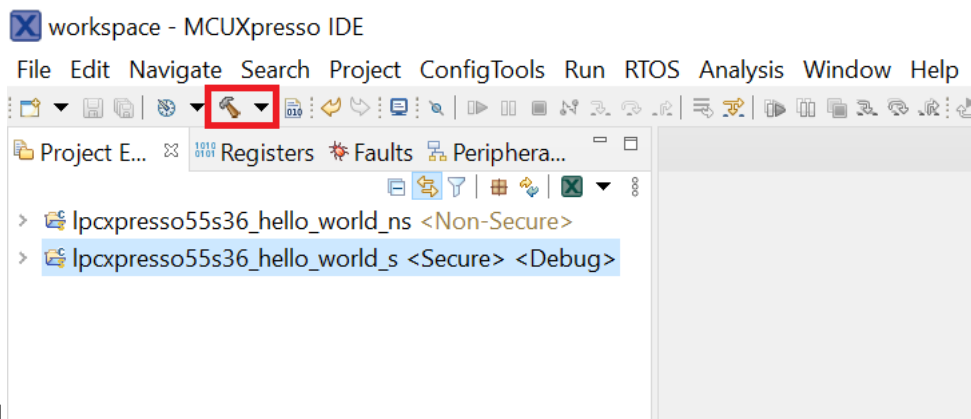
1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for LPCXpresso55S36 is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **LPC55xx** folder and select **LPCXpresso55S36**. Then, select **lpcxpresso55s36** and click **Next**.



2. Expand the trustzone_examples/ folder and select hello_world_s. Because TrustZone examples are linked together, the non-secure project is automatically imported with the secure project, and there is no need to select it explicitly. Then select **UART** as SDK Debug Console. Then, click **Finish**.



3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the lpcpresso55s36_hello_world_s project (TrustZone 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 the **Debug** target.



The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when the secure project

(continues on next page)

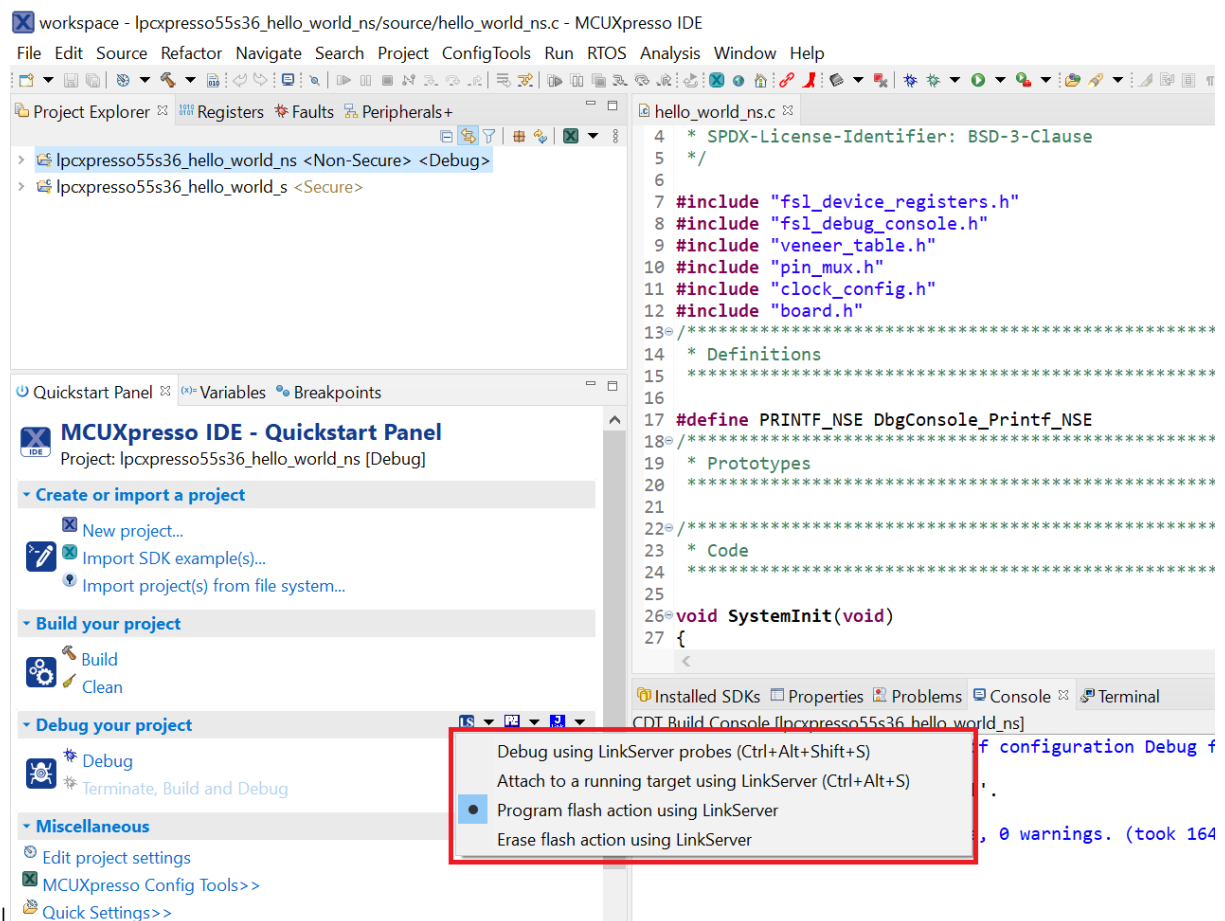
(continued from previous page)

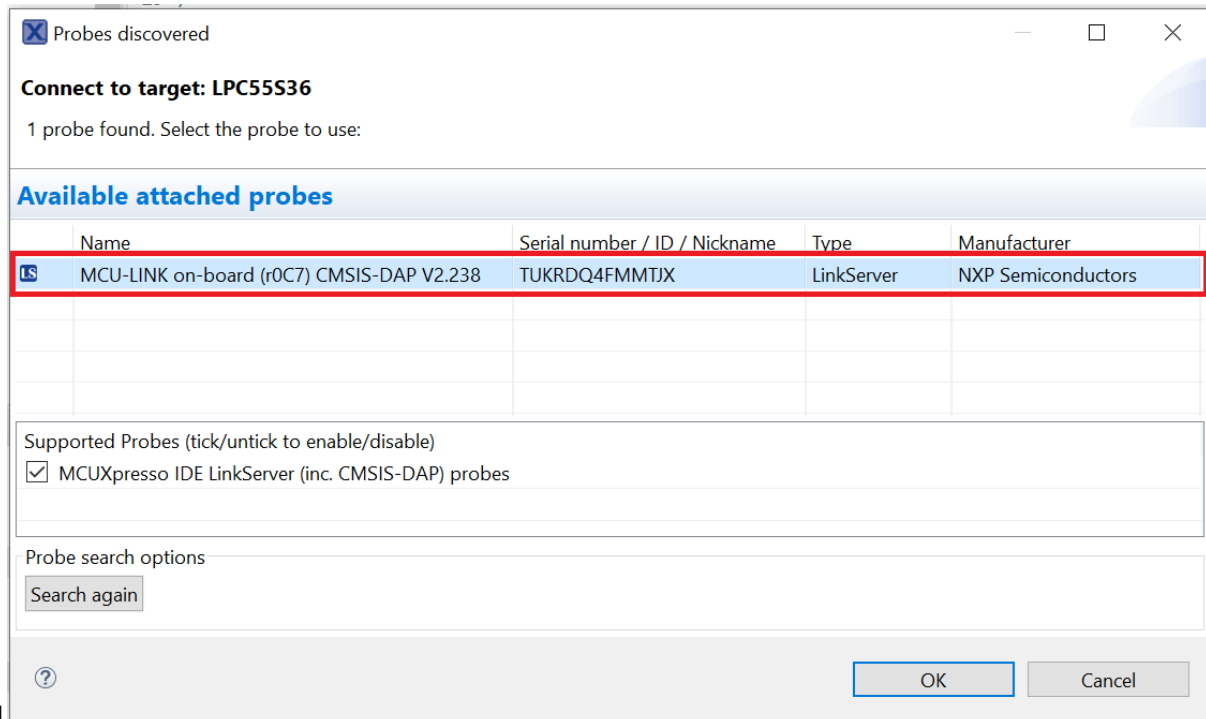
↪since CMSE library is not ready.

Note: When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click to select the second project. Right click in the Project Explorer view to display the context-sensitive menu and select **Build Configurations** \> **Set Active** \> **Release**. This is also possible by using the menu item of **Project** \> **Build Configuration** \> **Set Active** \> **Release**. After switching to the **Release** build configuration. Build the application for the secure project first.

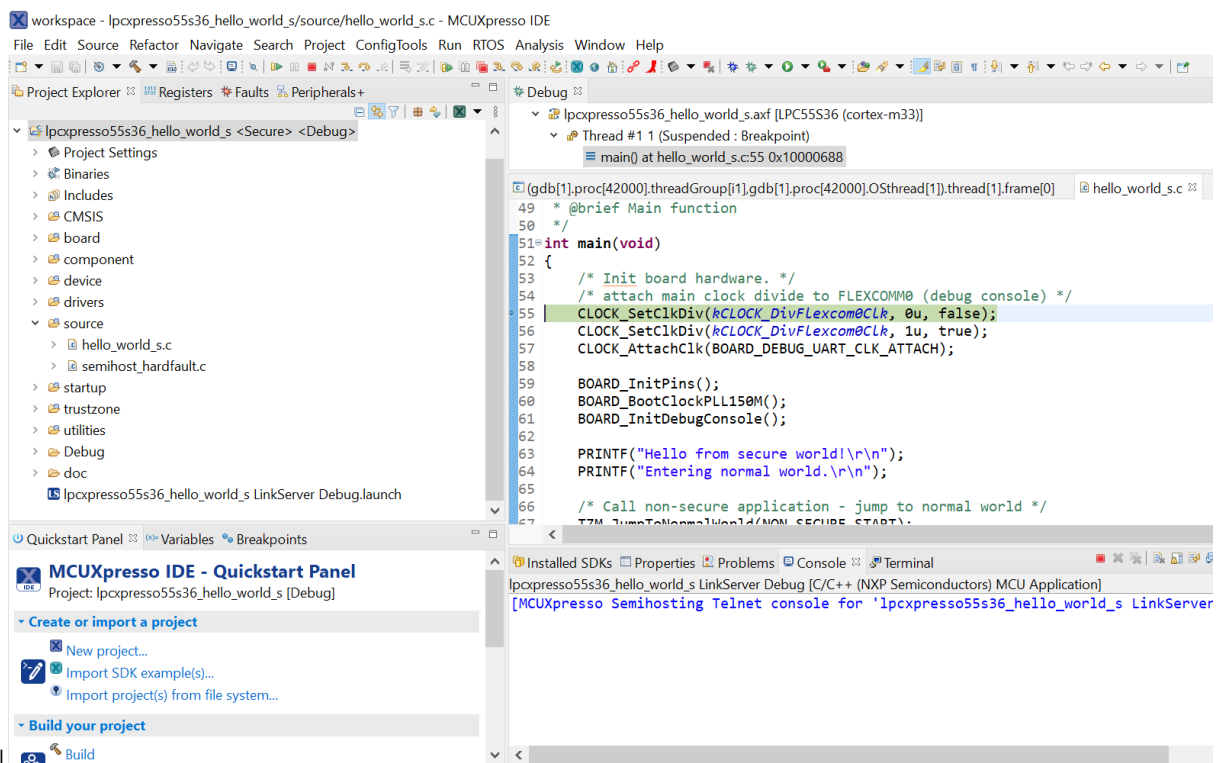
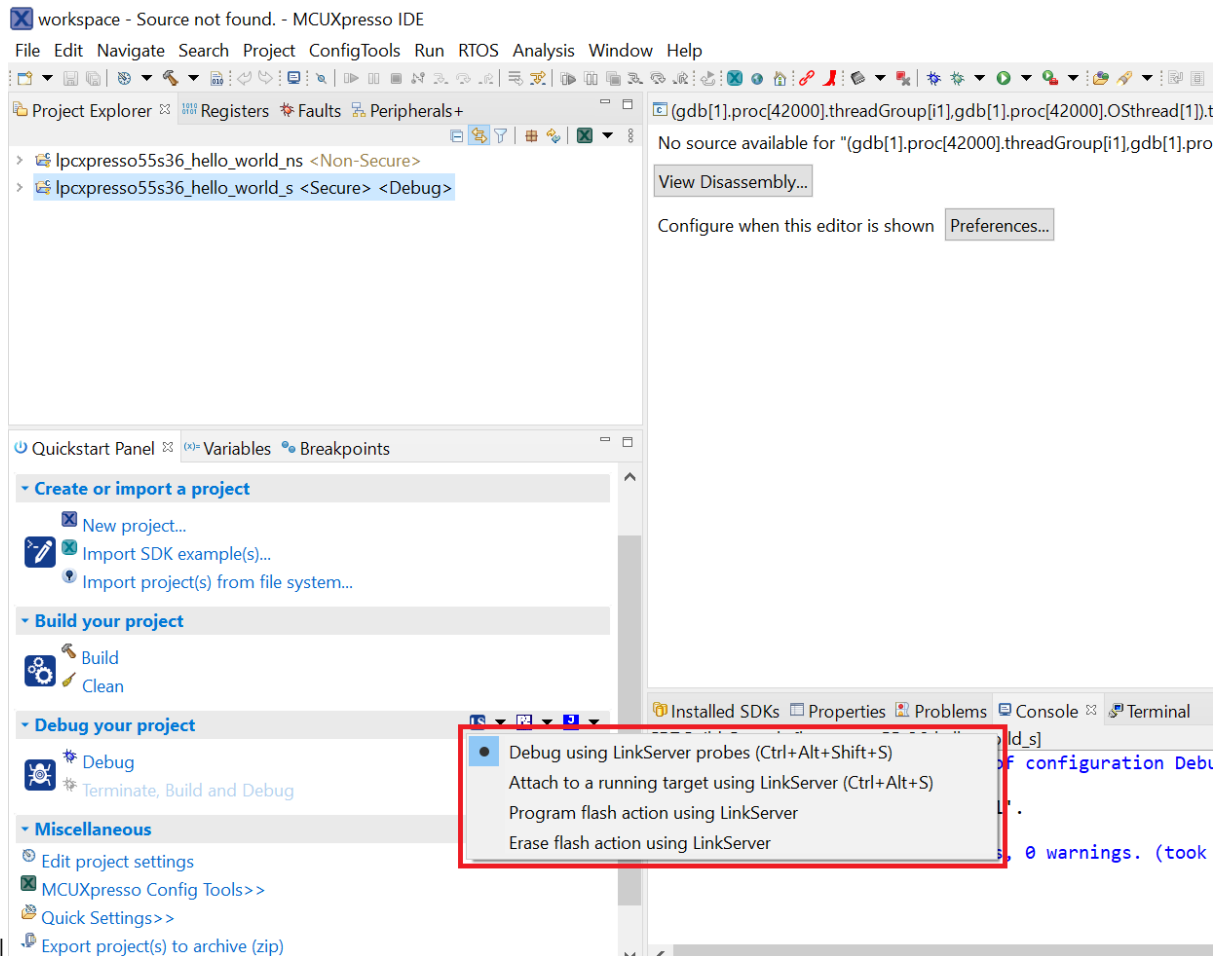
Parent topic: [Run a demo using MCUXpresso IDE](#)

Run a TrustZone example application To download and run the application perform all steps as described in *Section 3.3, "Run an example application"*. These steps are common for single core, dual-core, and TrustZone applications, ensuring both sides of the TrustZone application are properly loaded and started secure application. However, there is one additional dialogue that is specific to TrustZone examples. See the following figures as reference.

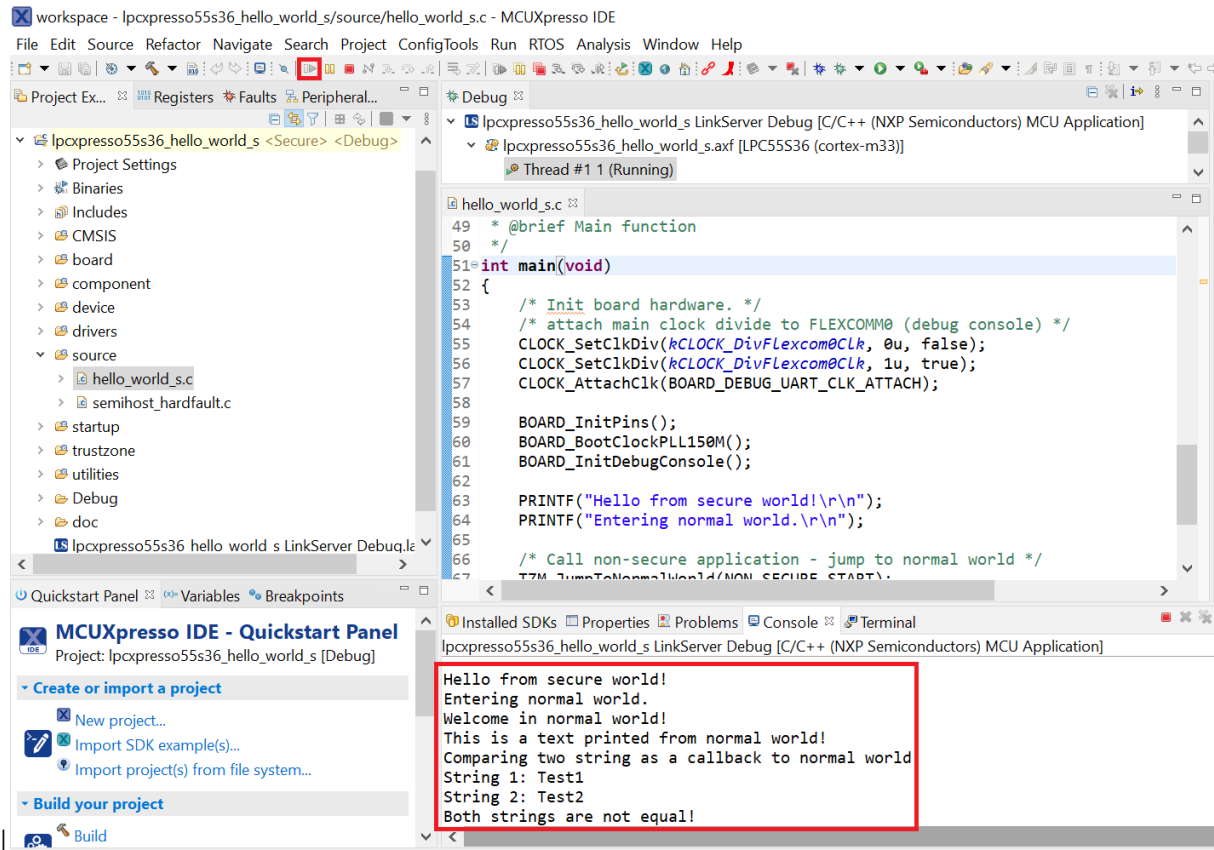




After loading the non-secure application, press **RESET** on board to release the device connect. Then, highlight the `lpcxpresso55s36_trustzone_examples_hello_world_s` project (TrustZone master project) in the Project Explorer. In the Quickstart Panel, click **lpcxpresso55s36_trustzone_examples_hello_world_s [Debug]** to launch the second debug session.



Start the application by clicking **Resume**. The `hello_world` TrustZone application then starts running, and the secure application starts the non-secure application during run time.



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

Run a demo application using IAR

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

Note: IAR Embedded Workbench for Arm version 8.32.3 is used in the following example, and the IAR toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes*.

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

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

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

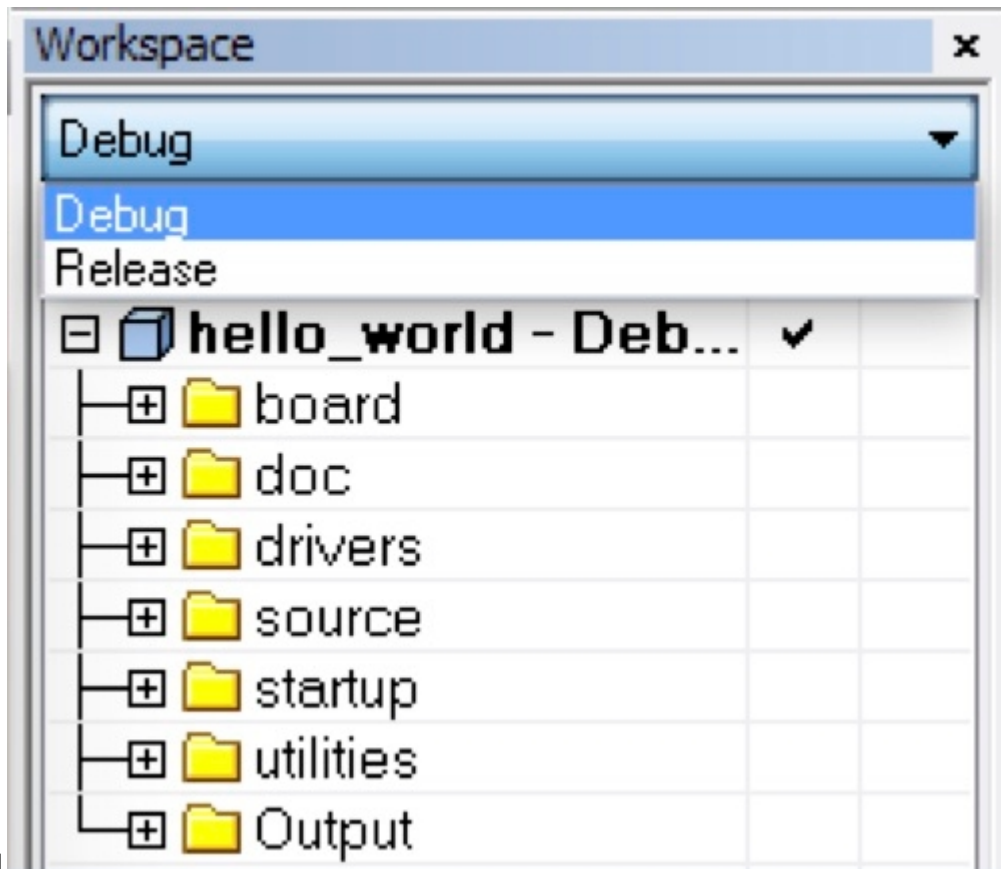
Using the LPCXpresso55S36 Freedom hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/LPCXpresso55S36/demo_apps/hello_world/iar/hello_world.eww
```

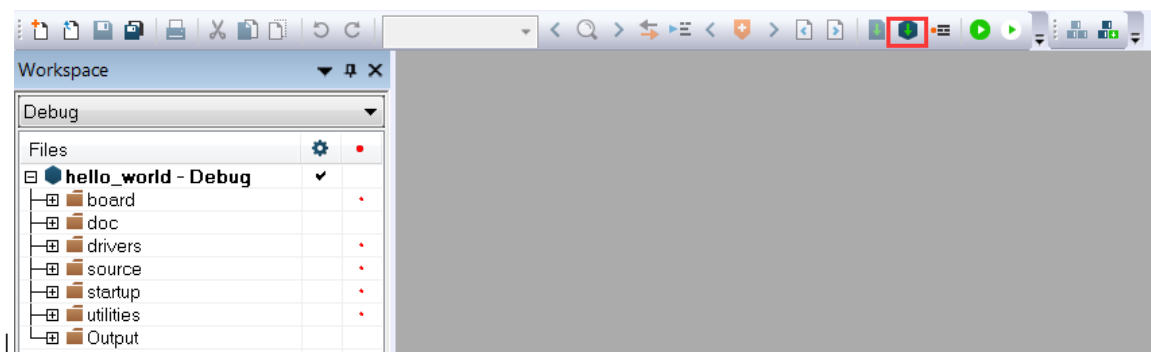
Other example applications may have additional folders in their path.

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

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



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



4. The build completes without errors.

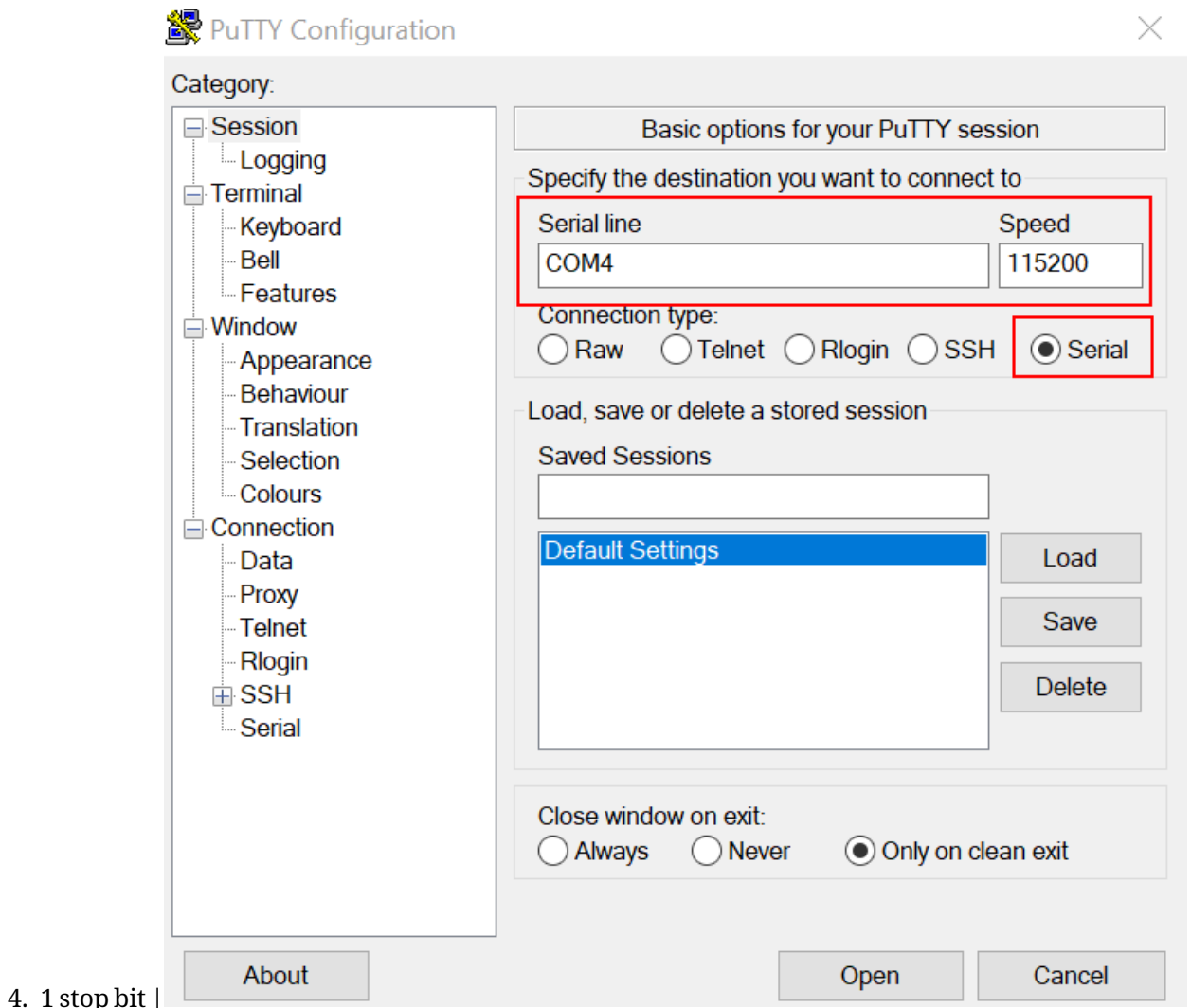
Parent topic: [Run a demo application using IAR](#)

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

1. See the table in [Default debug interfaces](#) to determine the debug interface that comes loaded on your specific hardware platform.
 - For boards with CMSIS-DAP/mbed/DAPLink interfaces, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the in-

structions to install the Windows® operating system serial driver. If running on Linux® OS, this step is not required.

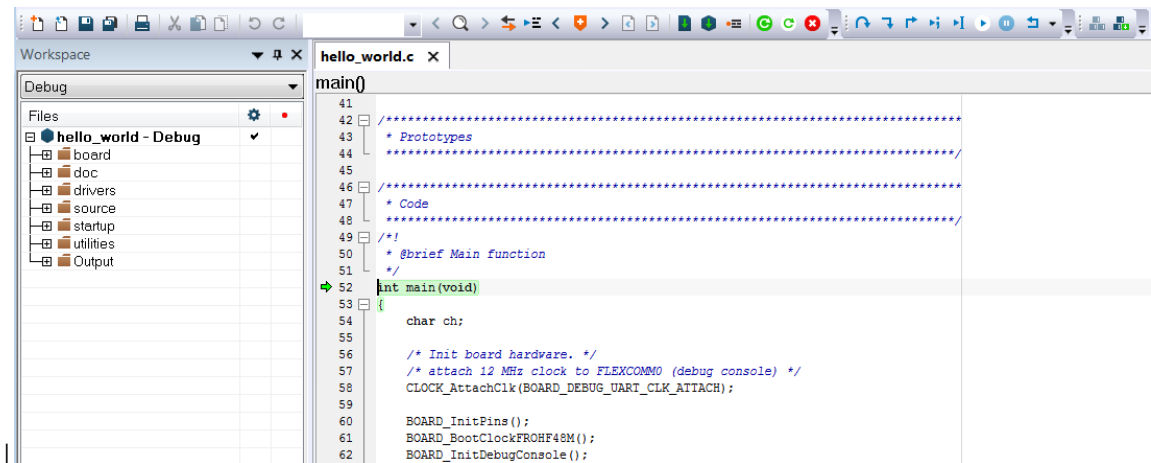
- For boards with P&E Micro interfaces, visit www.pemicro.com/support/downloads_find.cfm and download the P&E Micro Hardware Interface Drivers package.
2. Connect the development platform to your PC via USB cable.
 3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see [How to determine com port](#). Configure the terminal with the LPCXpresso55S36 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. In IAR, click the **Download and Debug** button to download the application to the target.



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



 when running debugging")

 when running debugging")

 when running debugging")

 when running debugging")

 when running debugging")

- Run the code by clicking the **Go** button.



- The `hello_world` application is now running and a banner is displayed on the terminal. If it does not appear, check your terminal settings and connections.



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

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

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/  
↪<application_name>_ns/iar
```

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/  
↪<application_name>_s/iar
```

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

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/cm33_core0/hello_world_ns/iar/  
↪hello_world_ns.eww
```

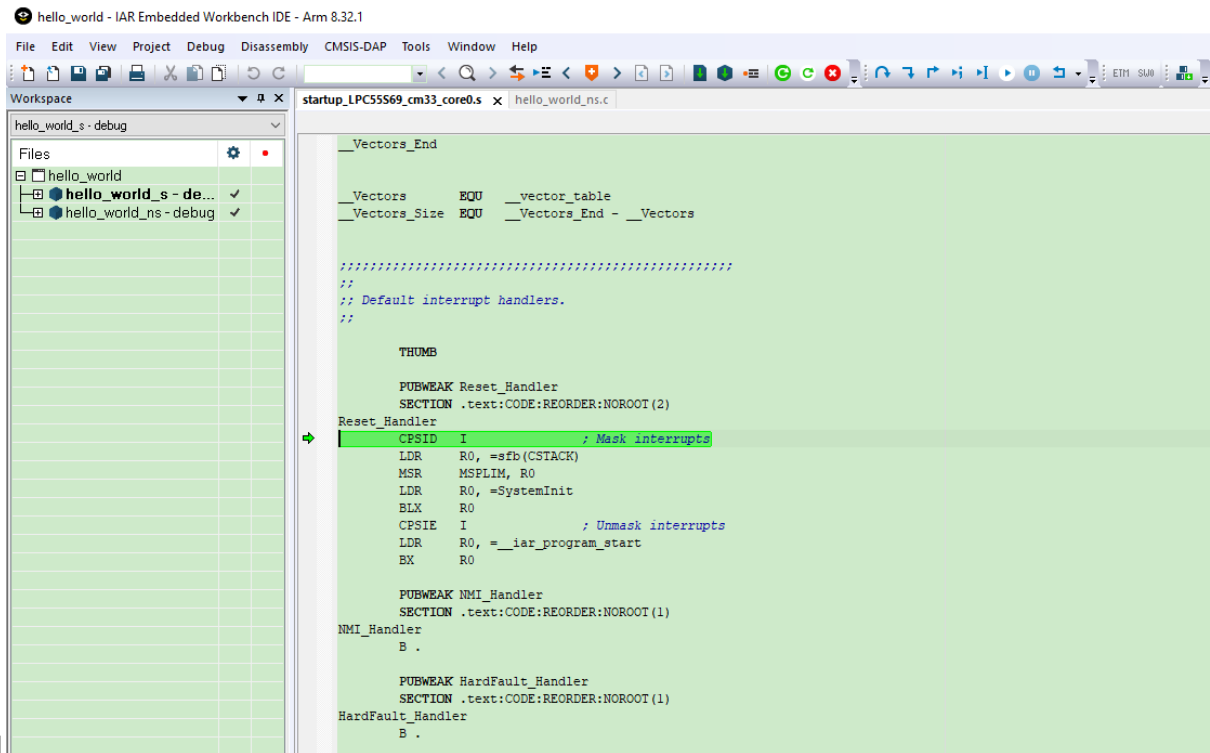
```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/cm33_core0/hello_world_s/iar/  
↪hello_world_s.eww
```

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/cm33_core0/hello_world_s/iar/  
↪hello_world.eww
```

This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project, since the CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project since CMSE library is not ready.

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

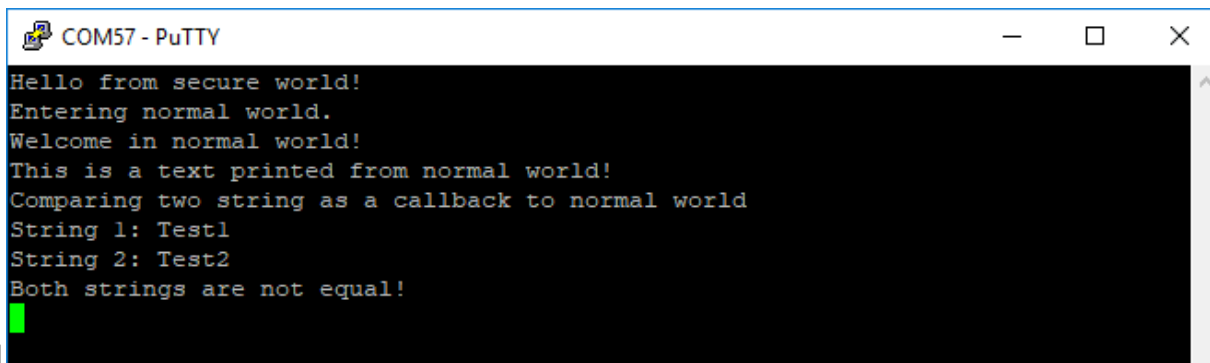
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files, so debugging can be fully managed from the secure project. To download and run the TrustZone application, switch to the secure application project and perform steps 1 – 4 as described in *Section 4.2, Run an example application*. These steps are common for both single core, dual-core, and TrustZone applications in IAR. After clicking **Download and Debug**, both the secure and non-secure image are loaded into the device flash memory, and the secure application is executed. It stops at the `Rest_Handler` function.



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



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



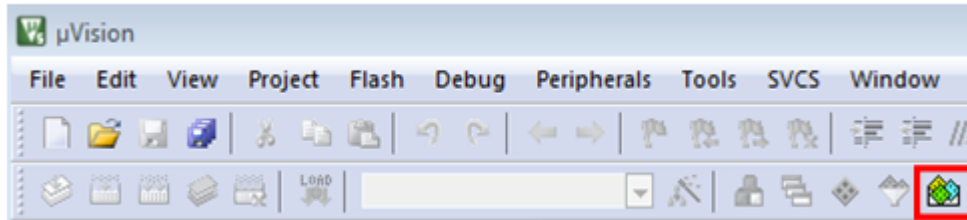
Parent topic: [Run a demo application using IAR](#)

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. The hello_world demo application targeted for the LPCXpresso55S36 hardware platform is used as an example, although these steps can be applied to any demo or example application 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 appropriate CMSIS pack.

1. Open the MDK IDE, which is called μ Vision. In the IDE, select the **Pack Installer** icon.



2. After the installation finishes, close the Pack Installer window and return to the μ Vision IDE.

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/LPCXpresso55S36/demo_apps/hello_world/mdk/hello_world.uvmpw
```

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



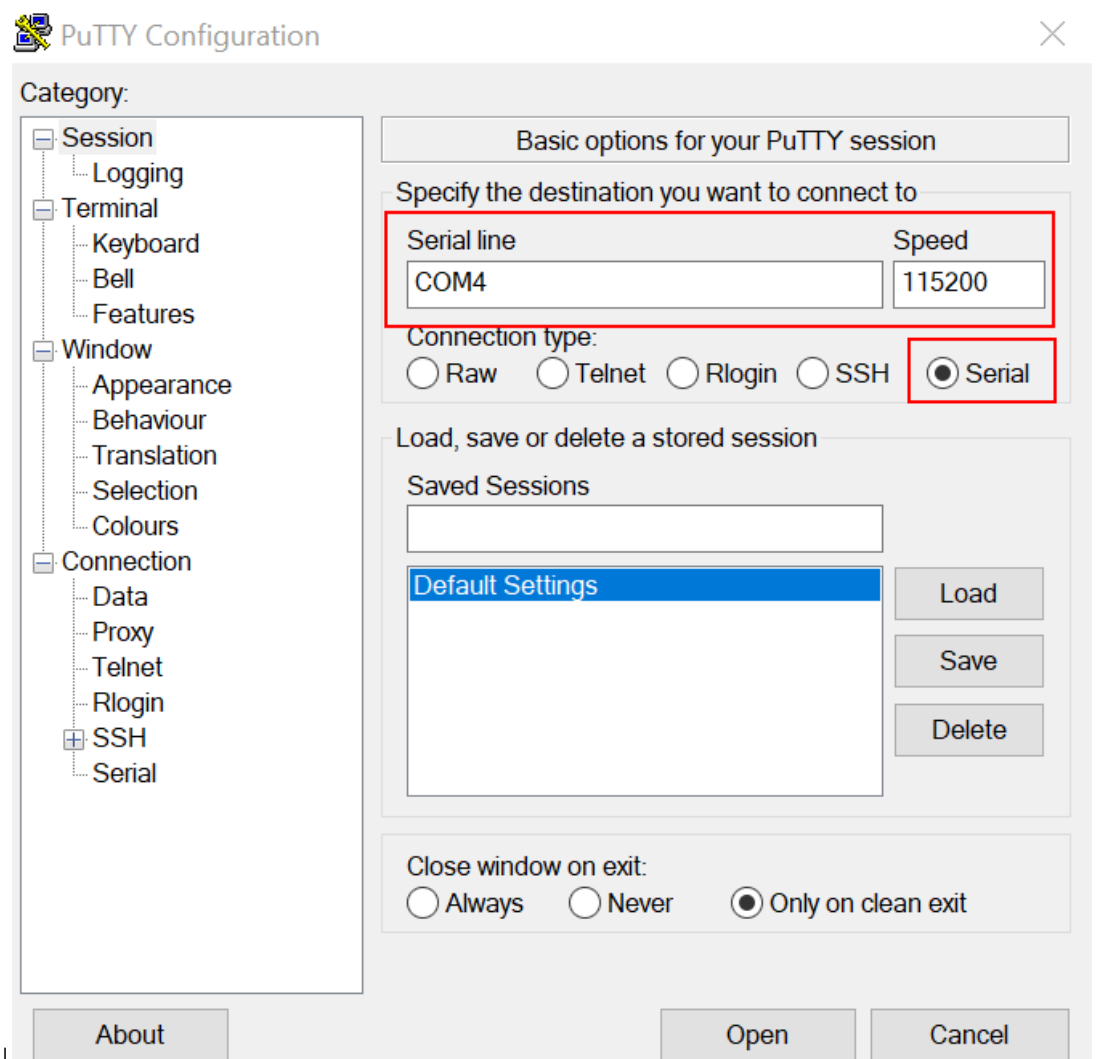
3. The build completes without errors.

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

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

1. See the table in [Default debug interfaces](#) to determine the debug interface that comes loaded on your specific hardware platform.
 - For boards with CMSIS-DAP/mbd/DAPLink interfaces, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows® operating system serial driver. If running on Linux® OS, this step is not required.
 - For boards with P&E Micro interfaces, visit www.pemicro.com/support/downloads_find.cfm and download the P&E Micro Hardware Interface Drivers package.
 - If using J-Link either a standalone debug pod or OpenSDA, install J-Link software (drivers and utilities) from <https://www.segger.com/downloads/jlink/>.

- If using J-Link either a standalone debug pod or J-link firmware programmed into the on-board debug probe, install the J-Link software (drivers and utilities) from <https://www.segger.com/downloads/jlink/>.
 - For boards with the OSJTAG interface, install the driver from <https://www.keil.com/download/>.
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 COM port (to determine the COM port number, see [How to determine com port](#). Configure the terminal with the LPCXpresso55S36 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

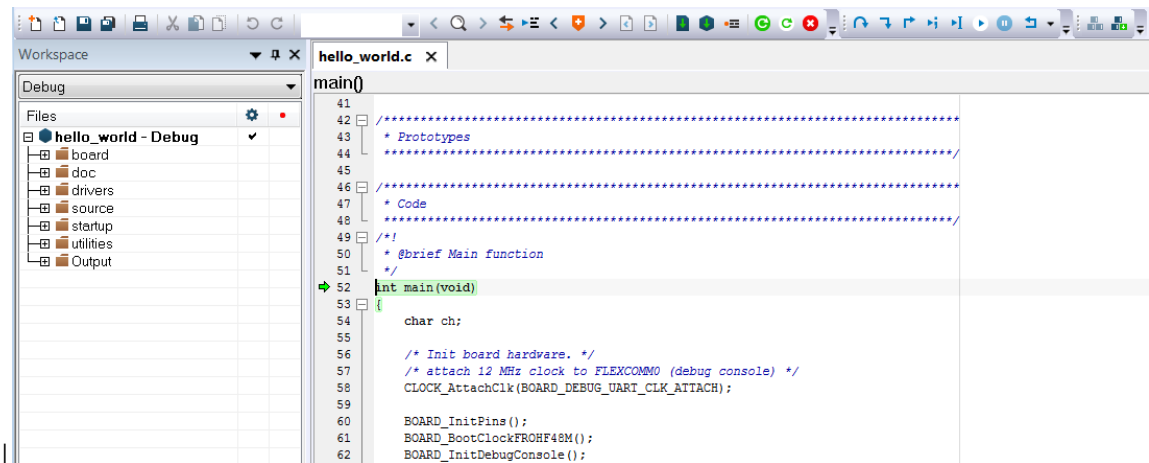


4. 1 stop bit

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



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



 when running debugging")

 when running debugging")

 when running debugging")

 when running debugging")

 when running debugging")

6. Run the code by clicking the **Go** button.



7. The `hello_world` application is now running and a banner is displayed on the terminal. If it does not appear, check your terminal settings and connections.



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

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

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World Keil MSDK/μVision ® workspaces are located in this folder:

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_ns.  
↪ eww
```

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.s.  
↪ uvmpw
```

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/hello_world_s/iar/hello_world.eww
```

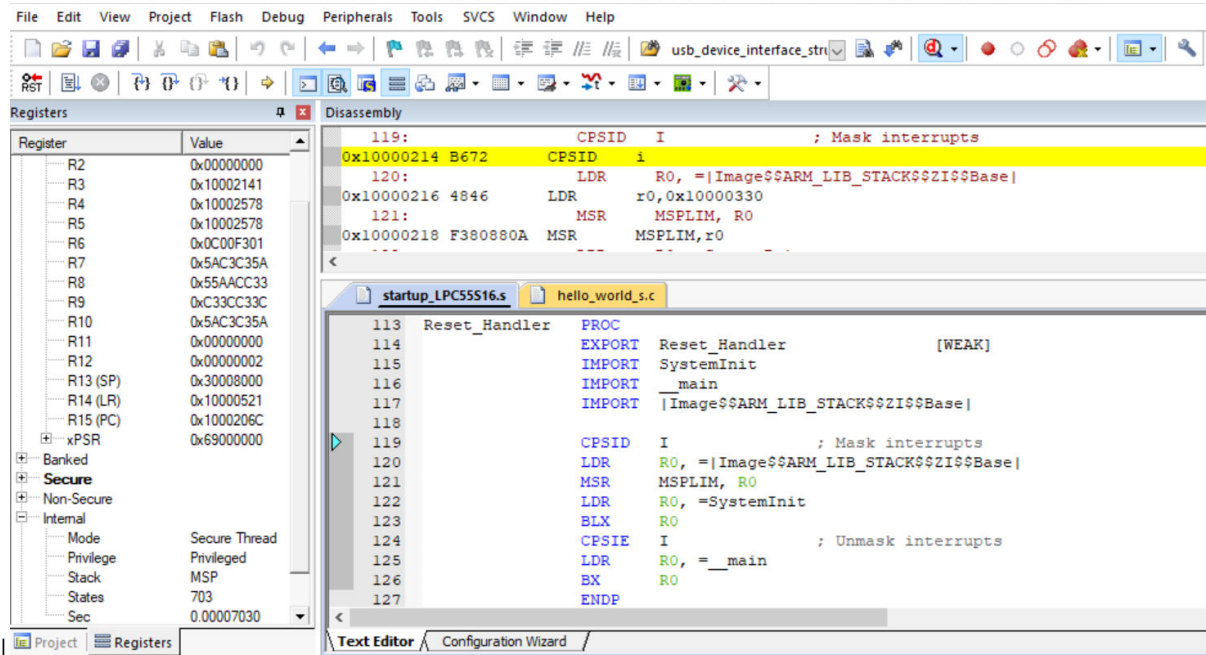
This project `hello_world.uvmpw` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another.

Build both applications separately by clicking **Rebuild**. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because CMSE library is not ready.

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

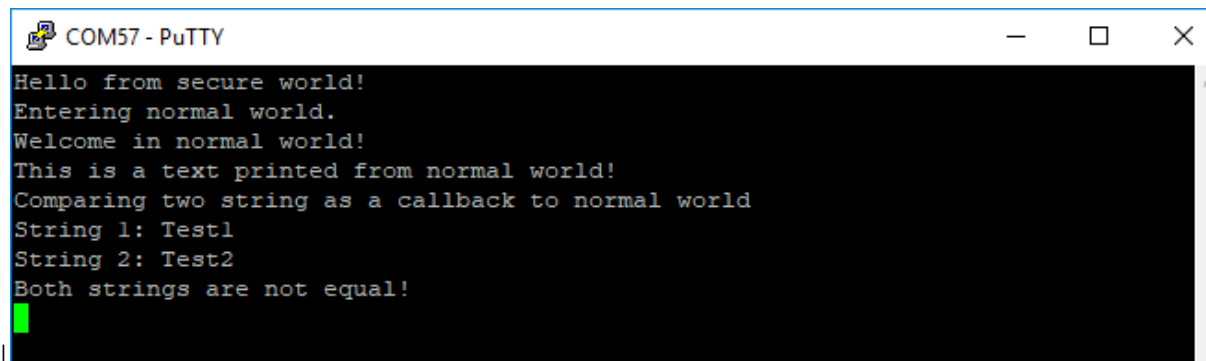
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

To download and run the TrustZone application, switch to the secure application project and perform steps as described in Run a TrustZone example application. These steps are common for single core, dual-core, and TrustZone applications in μVision. After clicking **Download and Debug**, both the secure and non-secure image are loaded into the device flash memory, and the secure application is executed. It stops at the function.



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

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 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_world demo application is targeted for the LPCXpresso55S36 hardware platform which is used as an example.

Note: Arm GCC version 7-2018-q2 is used as an example in this document. The latest GCC version for this package is as described in the *MCUXpresso SDK Release Notes for LPCXpresso55S36* (document MCUXSDKLPC553XRN).

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, and so on). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes*.

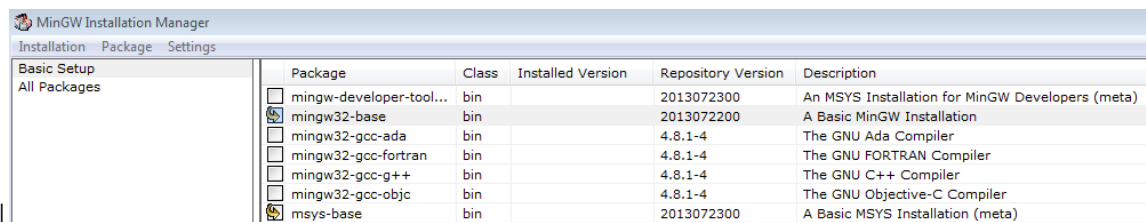
Parent topic:Set up toolchain

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.

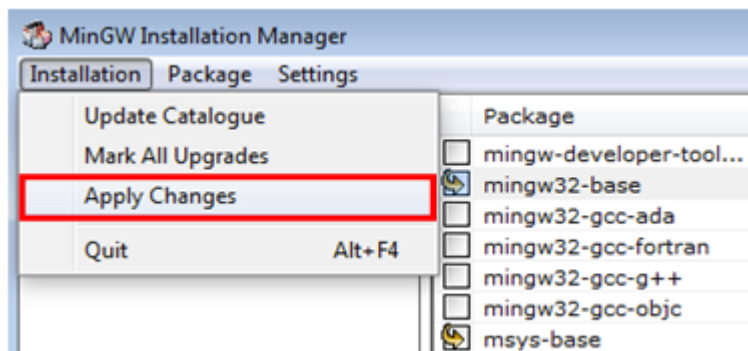
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is C:\MinGW, however, you may install to any location.

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.

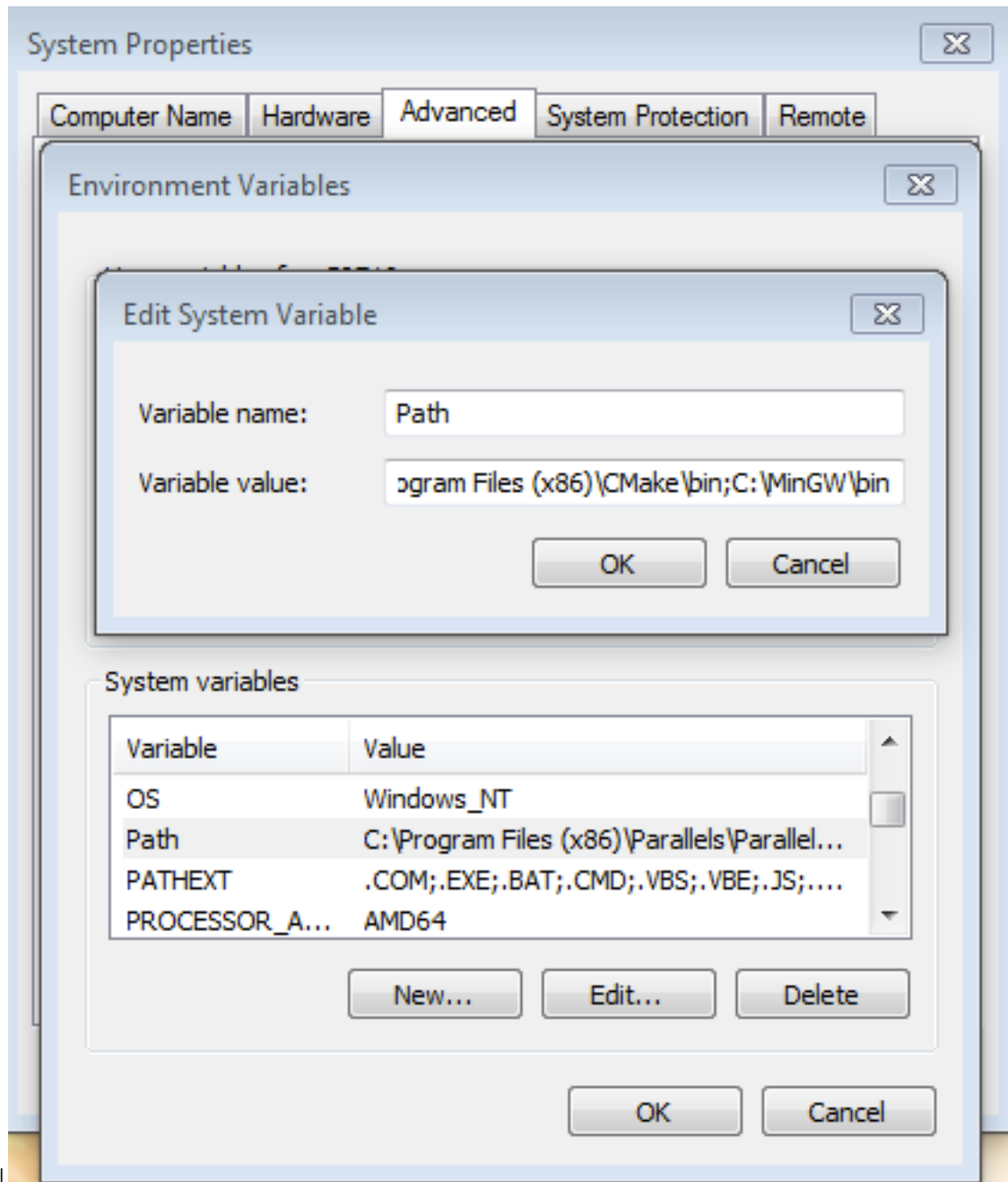


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:

```
<mingw_install_dir>\bin
```

Assuming the default installation path, C:\MinGW, an example is shown below. 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

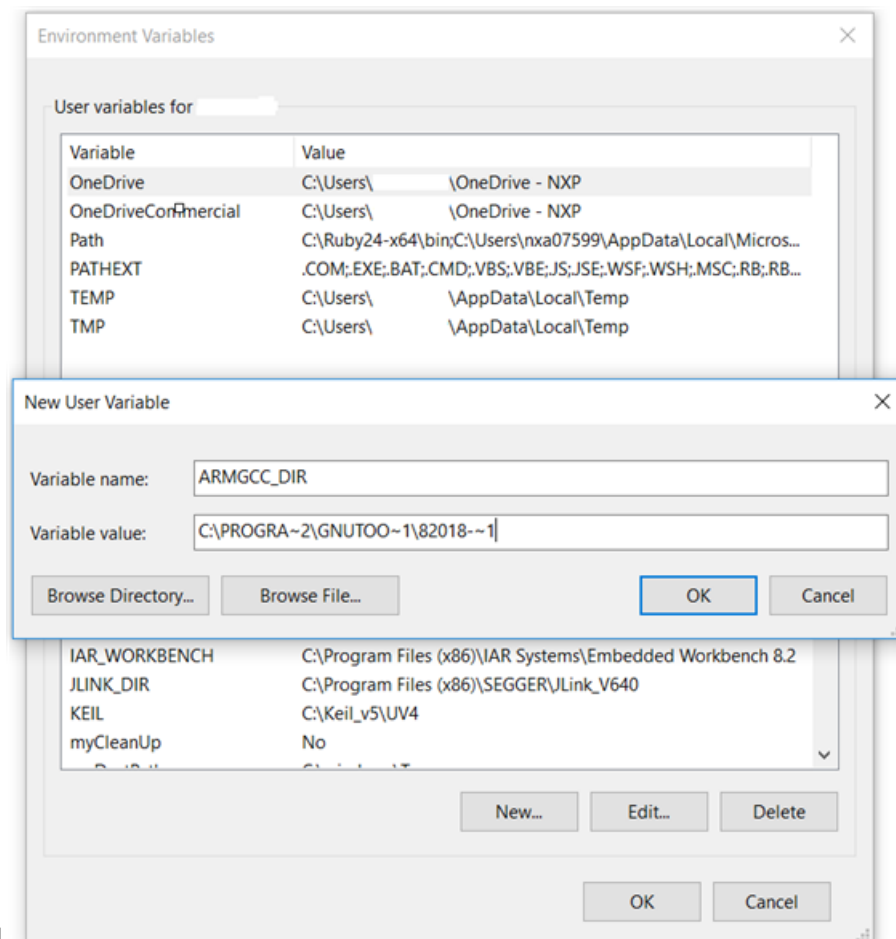
Add a new system environment variable for ARMGCC_DIR Create a new *system* environment variable and name it as ARMGCC_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

```
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 command for %I in (.) do echo %~sI

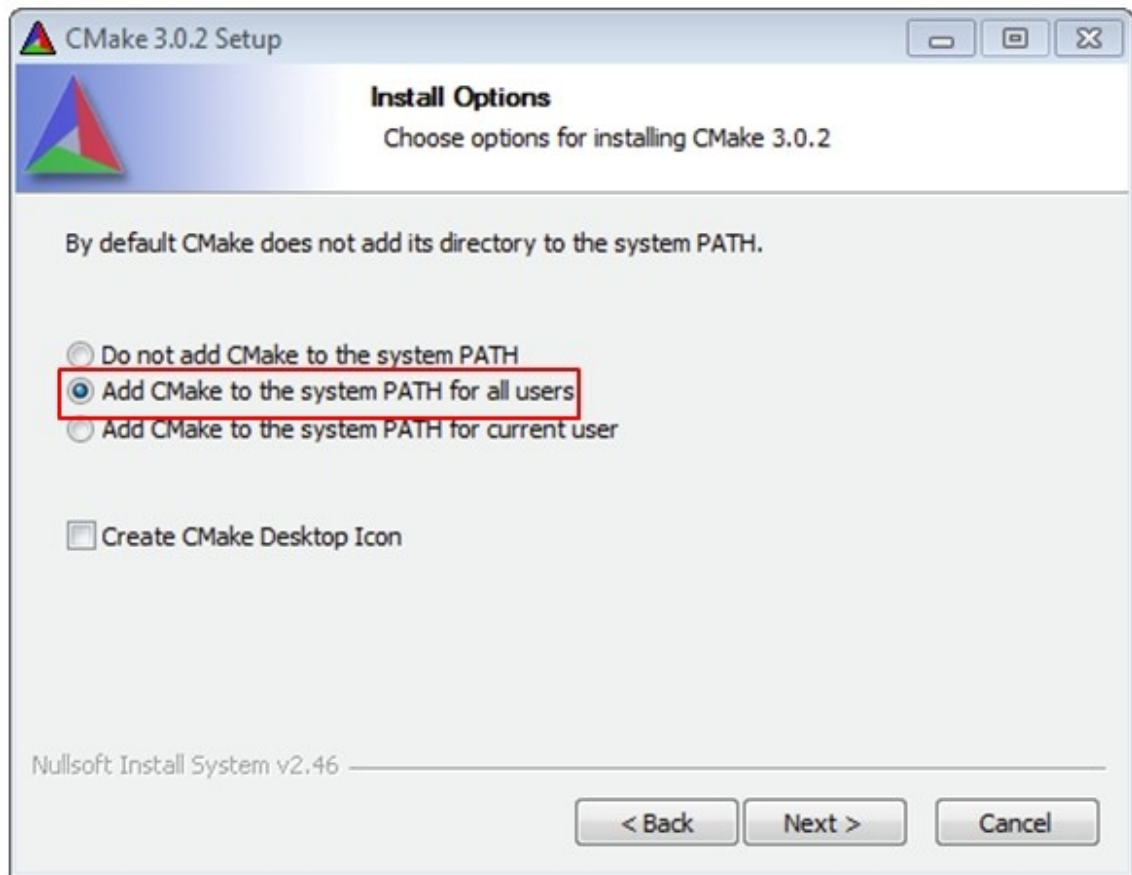
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```



Parent topic:Set up toolchain

Install CMake

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



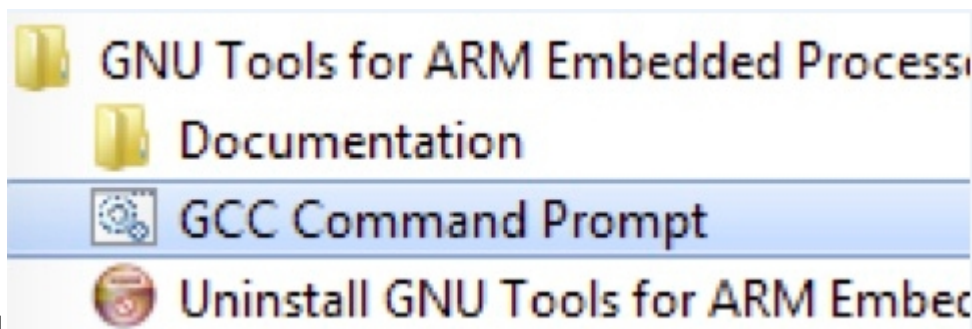
3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.
5. Make sure `sh.exe` is not in the Environment Variable PATH. This is a limitation of `mingw32-make`.

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>/<application_name>/armgcc
```

For this example, the exact path is:

```
<install_dir>/examples/lpcxpresso55s36/demo_apps/hello_world/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.

```
[ 95%] Building C object CMakeFiles/hello_world.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC5536/utilities/fsl_sbrk.c.obj
[100%] Linking C executable debug\hello_world.elf
Memory region      Used Size  Region Size  %age Used
  m_interrupts:      544 B        1 KB       53.13%
    m_text:        13528 B       255 KB       5.18%
    m_data:         3304 B       112 KB       2.88%
    m_sramx:           0 B        16 KB       0.00%
[100%] Built target hello_world.elf
C:\Users\nxf75278\Downloads\board_LPCXpresso55S36\boards\lpcxpresso55s36\demo_apps\hello_world\armgcc>pause
Press any key to continue . . .
```

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 complete the set-up check if your board supports OpenSDA in , see [Default debug interfaces](#).

If your board supports OpenSDA

- The OpenSDA interface on your board is pre-programmed with the J-Link OpenSDA firmware.
- For instructions on reprogramming the OpenSDA interface.

If your board does not support OpenSDA

- A standalone J-Link pod is required which should be connected to the debug interface of your board.

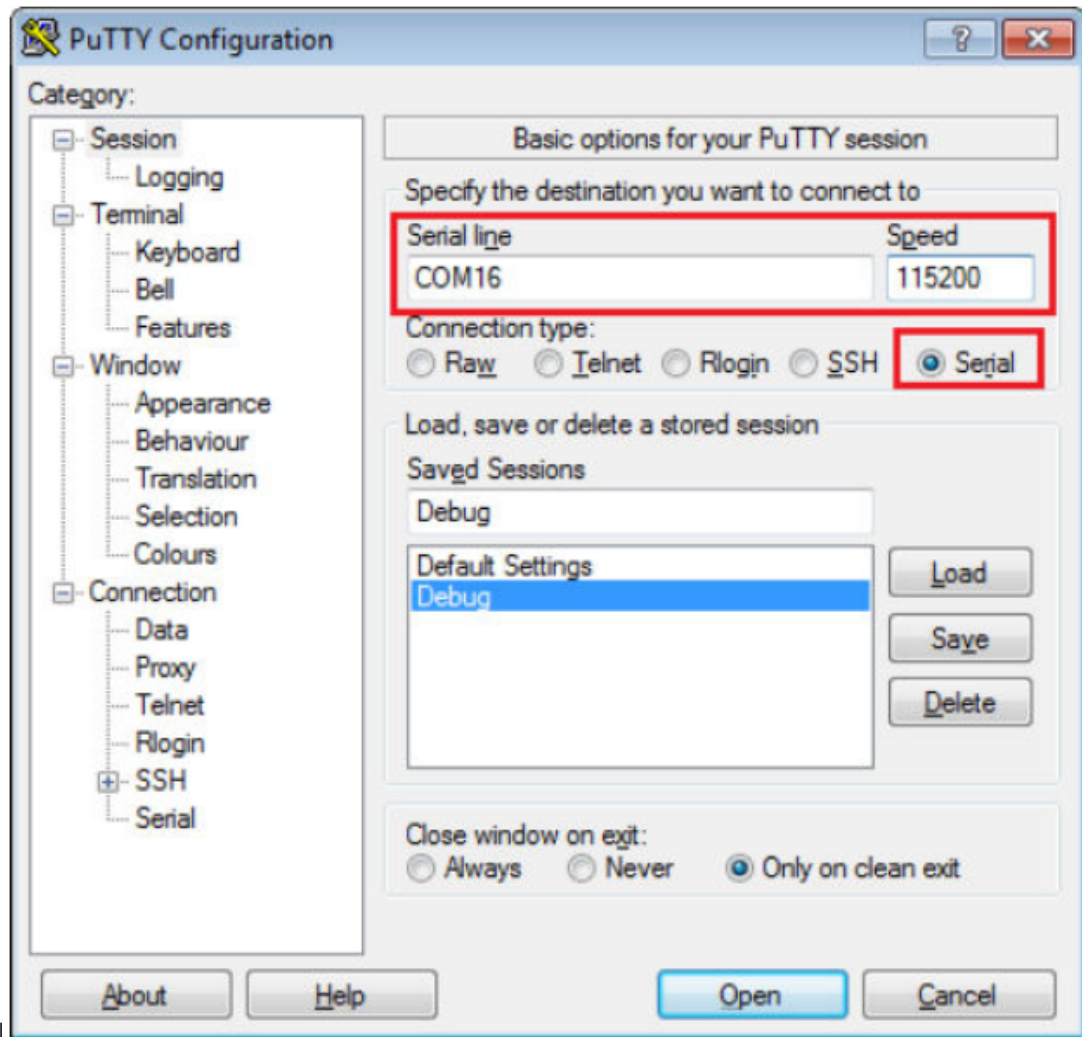
Note: Some hardware platforms require hardware modification in order to function correctly with an external debug interface.

Note: J-Link GDB Server application is not supported for TFM examples. Use CMSIS DAP instead of J-Link for flashing and debugging TFM examples.

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

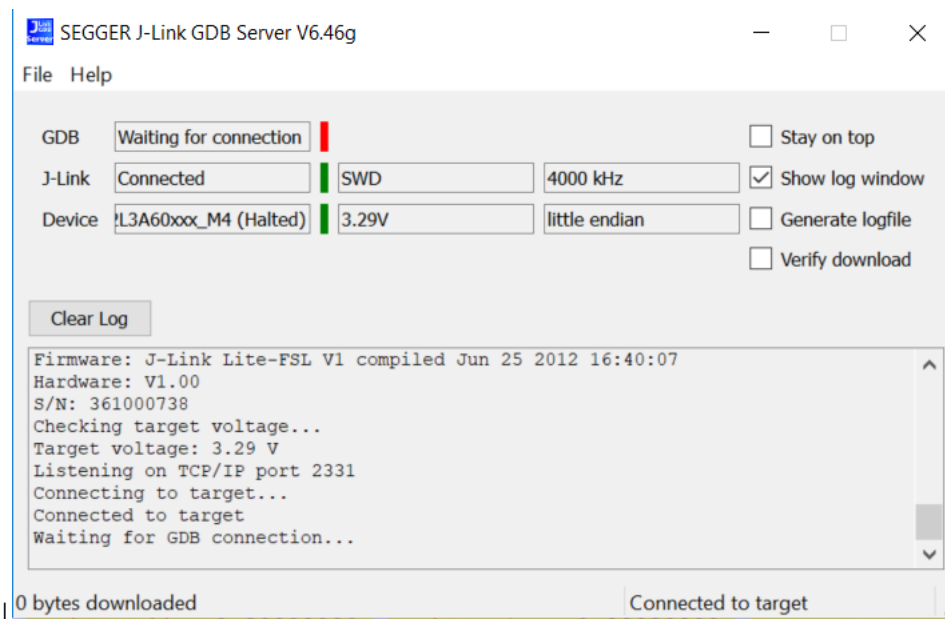
1. Connect the development platform to your PC via USB cable between the LPC-Link2 USB connector (may be named OSJTAG for some boards) and the PC USB connector. If using a standalone J-Link debug pod, connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see [How to determine com port](#)).
3. Configure the terminal with these settings:
 1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)
 2. No parity

3. 8 data bits
4. 1 stop bit

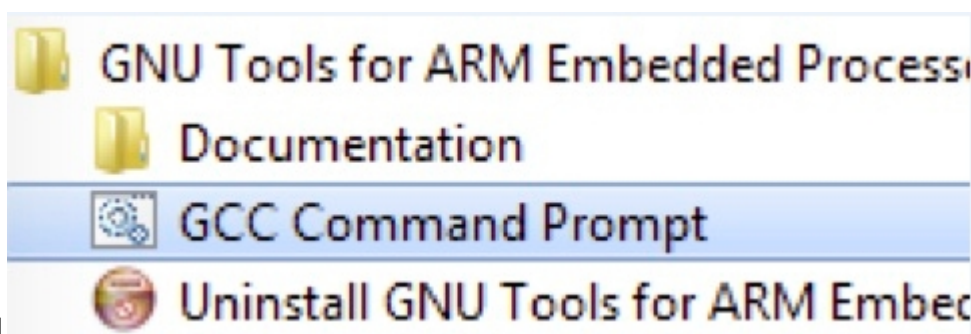


****Note:**** Make sure the board is set to FlexSPI flash boot mode \(\ISP2: ISP1: ISP0 = ON, OFF, ON\) before use GDB debug.

4. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system Start menu and selecting **Programs -> SEGGER -> J-Link <version> J-Link GDB Server**.
5. Modify the settings as shown below. The target device selection chosen for this example is LPC55S36
6. After it is connected, the screen should look like this figure:



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>/armgcc/debug
```

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

For this example, the path is:

```
<install_dir>/boards/lpcxpresso55s36/demo_apps/hello_world/cm4/armgcc/debug
```

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

```

MINGW64: c:/Users/nxf75278/Downloads/board_LPCXpresso55S36/boards/lpcxpresso55s36/demo_apps/hello_world/armgcc/debug
nxf75278@NXL53550 MINGW64 ~:/Downloads/board_LPCXpresso55S36/boards/lpcxpresso55s36/demo_apps/hello_world/armgcc/debug
$ arm-none-eabi-gdb.exe hello_world.elf
GNU gdb (GNU Arm Embedded Toolchain 10.3-2021.07) 10.2.90.20210621-git
Copyright (c) 2021 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:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world.elf...
(gdb)

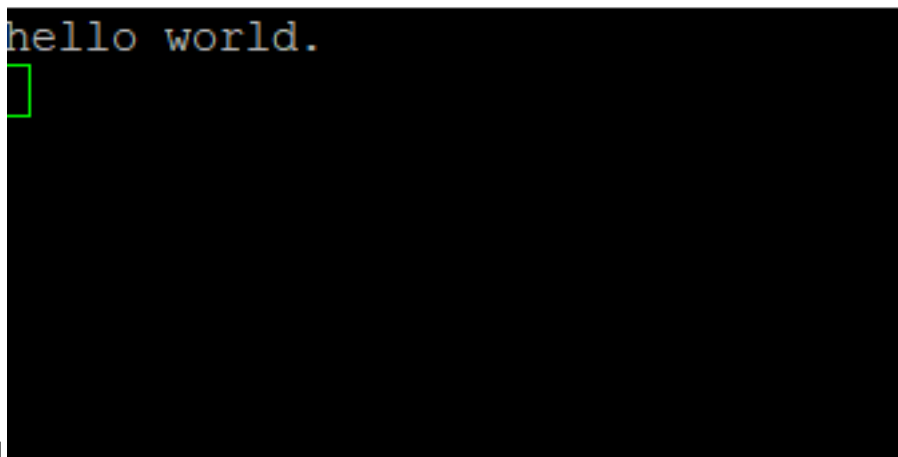
```

10. Run these commands:

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

11. The application is now downloaded and halted at the watch point. Execute the monitor go command to start the demo application.

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



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

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

```

<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/
→ <application_name>_ns/armgcc

```

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/  
↪<application_name>_s/armgcc
```

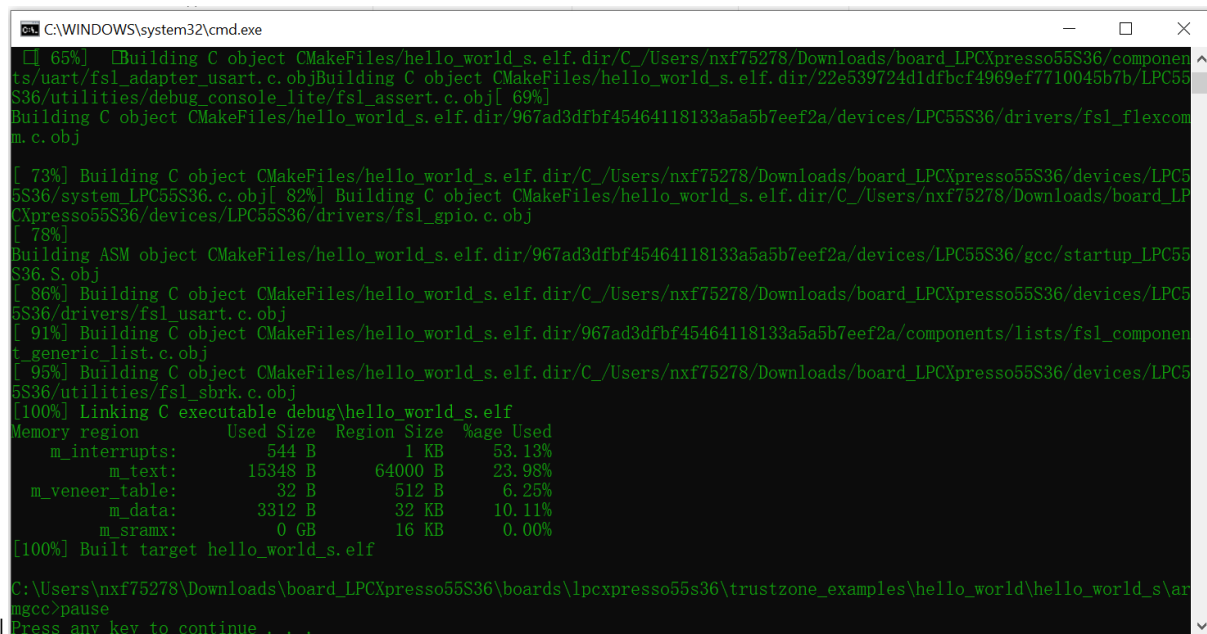
Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World GCC build scripts are located in this folder:

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_ns.  
↪eww
```

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/hello_world_s/iar/hello_world_s.  
↪eww
```

```
<install_dir>/boards/lpcxpresso55s36/trustzone_examples/hello_world/hello_world_s/iar/hello_world.eww
```

Build both applications separately, following steps for single core examples as described in *Section 6.2, “Build an example application”*. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project, since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because the CMSE library is not ready.



```
C:\WINDOWS\system32\cmd.exe
[ 65%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/components/uart/fsl_adapter_usart.c.obj
Building C object CMakeFiles/hello_world_s.elf.dir/22e539724d1dfbcf4969ef7710045b7b/LPC55S36/utilities/debug_console_lite/fsl_assert.c.obj[ 69%]
Building C object CMakeFiles/hello_world_s.elf.dir/967ad3dfbf45464118133a5a5b7eef2a/devices/LPC55S36/drivers/fsl_flexcom_m.c.obj
[ 73%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/system_LPC55S36.c.obj[ 82%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/drivers/fsl_gpio.c.obj
[ 78%] Building ASM object CMakeFiles/hello_world_s.elf.dir/967ad3dfbf45464118133a5a5b7eef2a/devices/LPC55S36/gcc/startup_LPC55S36.S.obj
[ 86%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/drivers/fsl_usart.c.obj
[ 91%] Building C object CMakeFiles/hello_world_s.elf.dir/967ad3dfbf45464118133a5a5b7eef2a/components/lists/fsl_component_generic_list.c.obj
[ 95%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/utilities/fsl_sbrk.c.obj
[100%] Linking C executable debug\hello_world_s.elf
Memory region      Used Size   Region Size  %age Used
  m_interrupts:      544 B         1 KB       53.13%
    m_text:        15348 B       64000 B       23.98%
  m_veneer_table:     32 B          512 B        6.25%
    m_data:         3312 B         32 KB       10.11%
    m_sramx:           0 B          16 KB        0.00%
[100%] Built target hello_world_s.elf
C:\Users\nxf75278\Downloads\board_LPCXpresso55S36\boards\lpcxpresso55s36\trustzone_examples\hello_world\hello_world_s\armgcc>pause
Press any key to continue . . .
```

```

C:\WINDOWS\system32\cmd.exe
Building C object CMakeFiles/hello_world_ns.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/build/CMakeFiles/hello_world_ns.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/drivers/fsl_power.c.obj
[ 60%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/drivers/fsl_clock.c.obj [ 65%] Building C object CMakeFiles/hello_world_ns.elf.dir/22e539724d1dfbcf4969ef7710045b7b/LPC55S36/utilities/debug_console_lite/fsl_assert.c.obj [ 70%] Building C object CMakeFiles/hello_world_ns.elf.dir/967ad3dfbf45464118133a5a5b7eef2a/devices/LPC55S36/drivers/fsl_flexcomm.c.obj [ 75%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/967ad3dfbf45464118133a5a5b7eef2a/devices/LPC55S36/gcc/startup_LPC55S36.S.obj [ 80%] [ 85%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/drivers/fsl_gpio.c.obj Building C object CMakeFiles/hello_world_ns.elf.dir/C:/Users/nxf75278/Downloads/board_LPCXpresso55S36/devices/LPC55S36/system_LPC55S36.c.obj [ 90%] Building C object CMakeFiles/hello_world_ns.elf.dir/967ad3dfbf45464118133a5a5b7eef2a/components/lists/fsl_component_generic_list.c.obj [ 95%] Building C object CMakeFiles/hello_world_ns.elf.dir/967ad3dfbf45464118133a5a5b7eef2a/devices/LPC55S36/utilities/fsl_sbrk.c.obj [100%] Linking C executable debug\hello_world_ns.elf
Memory region      Used Size  Region Size  %age Used
  m_interrupts:      544 B        1 KB       53.13%
    m_text:        4880 B       191 KB        2.50%
    m_data:        3288 B        80 KB        4.01%
    m_sramx:         0 GB        16 KB        0.00%
[100%] Built target hello_world_ns.elf
C:\Users\nxf75278\Downloads\board_LPCXpresso55S36\boards\lpcxpresso55s36\trustzone_examples\hello_world\hello_world_ns\armgcc>pause
Press any key to continue . . .

```

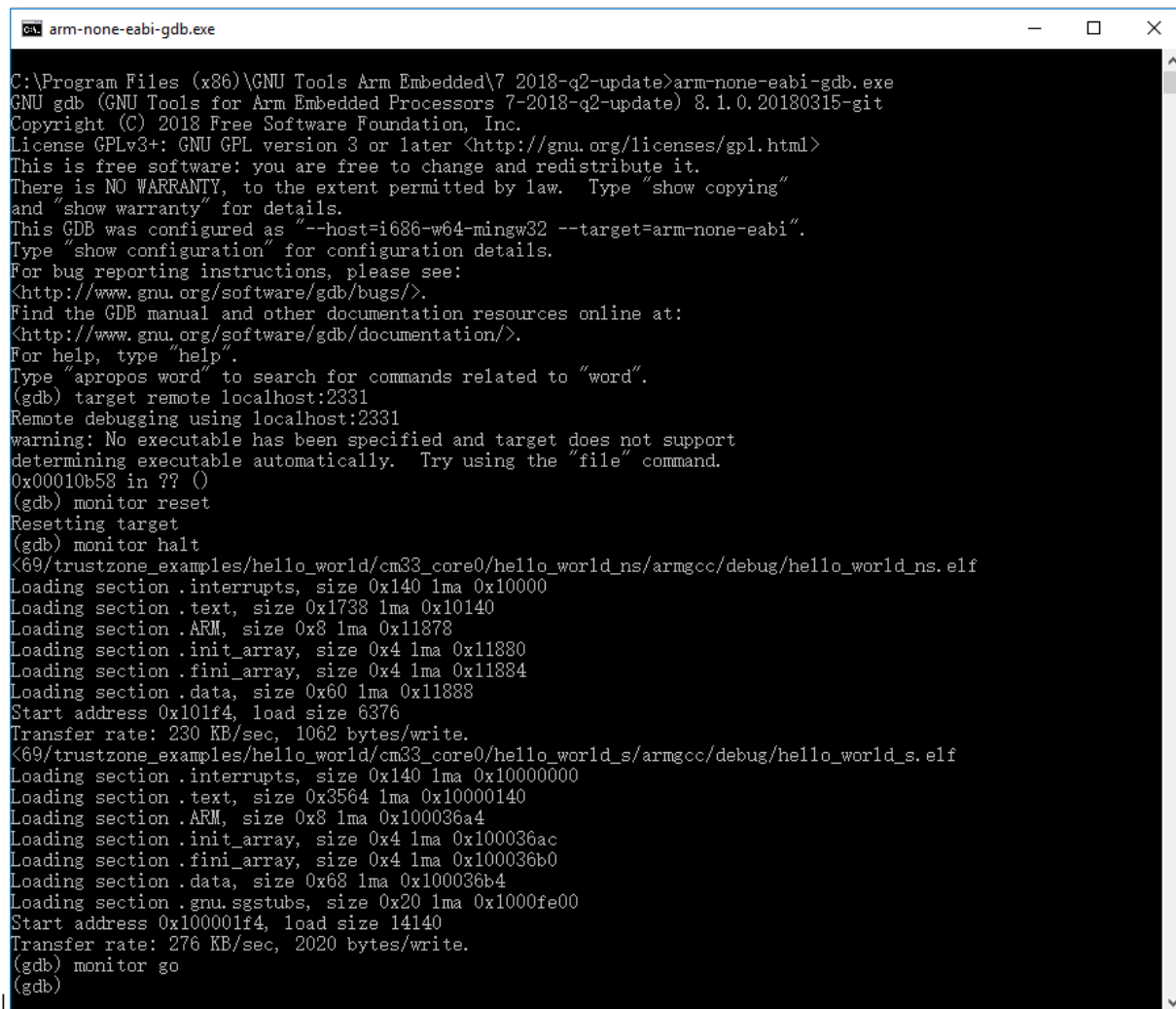
Parent topic: [Run a demo using Arm® GCC](#)

Run a TrustZone example application When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in [Section 6.3, “Run an example application”](#).

To download and run the TrustZone application, perform steps 1 to 10, as described in [Section 5.3, “Run an example application”](#). These steps are common for both single core and trustzone applications in Arm GCC.

Then, run these commands:

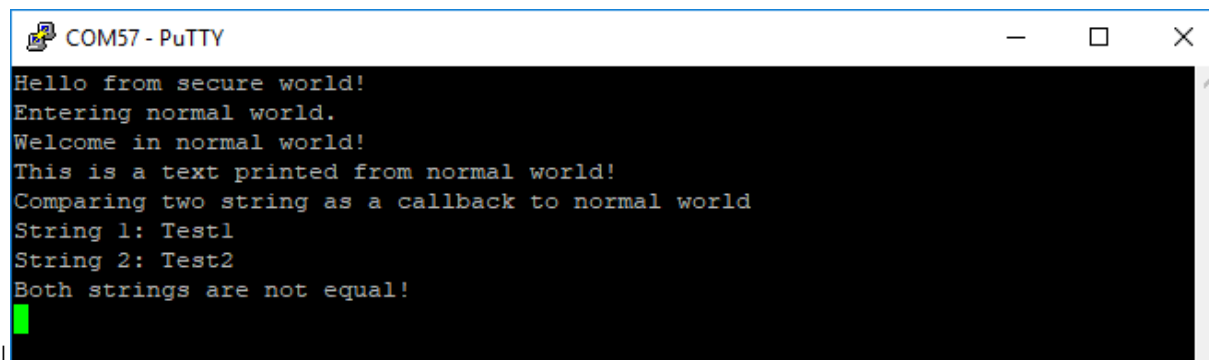
1. arm-none-eabi-gdb.exe
2. target remote localhost:2331
3. monitor reset
4. monitor halt
5. load <install_dir>/boards/lpcxpresso55s36/trustzone/_examples/hello/_world/cm33/_core0/hello/_world/_ns/armgcc/debug/hello/_world/_ns.elf
6. load <install_dir>/boards/lpcxpresso55s36/trustzone/_examples/hello/_world/cm33/_core0/hello/_world/_s/armgcc/debug/hello/_world/_s.elf
7. The application is now downloaded and halted at the watch point. Execute the monitor go command to start the demo application.



```

C:\Program Files (x86)\GNU Tools Arm Embedded\7 2018-q2-update>arm-none-eabi-gdb.exe
GNU gdb (GNU Tools for Arm Embedded Processors 7-2018-q2-update) 8.1.0.20180315-git
Copyright (C) 2018 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".
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x00010b58 in ?? ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
<69/trustzone_examples/hello_world/cm33_core0/hello_world_ns/armgcc/debug/hello_world_ns.elf
Loading section .interrupts, size 0x140 lma 0x10000
Loading section .text, size 0x1738 lma 0x10140
Loading section .ARM, size 0x8 lma 0x11878
Loading section .init_array, size 0x4 lma 0x11880
Loading section .fini_array, size 0x4 lma 0x11884
Loading section .data, size 0x60 lma 0x11888
Start address 0x101f4, load size 6376
Transfer rate: 230 KB/sec, 1062 bytes/write.
<69/trustzone_examples/hello_world/cm33_core0/hello_world_s/armgcc/debug/hello_world_s.elf
Loading section .interrupts, size 0x140 lma 0x10000000
Loading section .text, size 0x3564 lma 0x10000140
Loading section .ARM, size 0x8 lma 0x100036a4
Loading section .init_array, size 0x4 lma 0x100036ac
Loading section .fini_array, size 0x4 lma 0x100036b0
Loading section .data, size 0x68 lma 0x100036b4
Loading section .gnu.sgstubs, size 0x20 lma 0x1000fe00
Start address 0x100001f4, load size 14140
Transfer rate: 276 KB/sec, 2020 bytes/write.
(gdb) monitor go
(gdb)

```



```

Hello from secure world!
Entering normal world.
Welcome in normal world!
This is a text printed from normal world!
Comparing two string as a callback to normal world
String 1: Test1
String 2: Test2
Both strings are not equal!

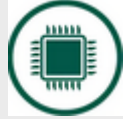
```

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:

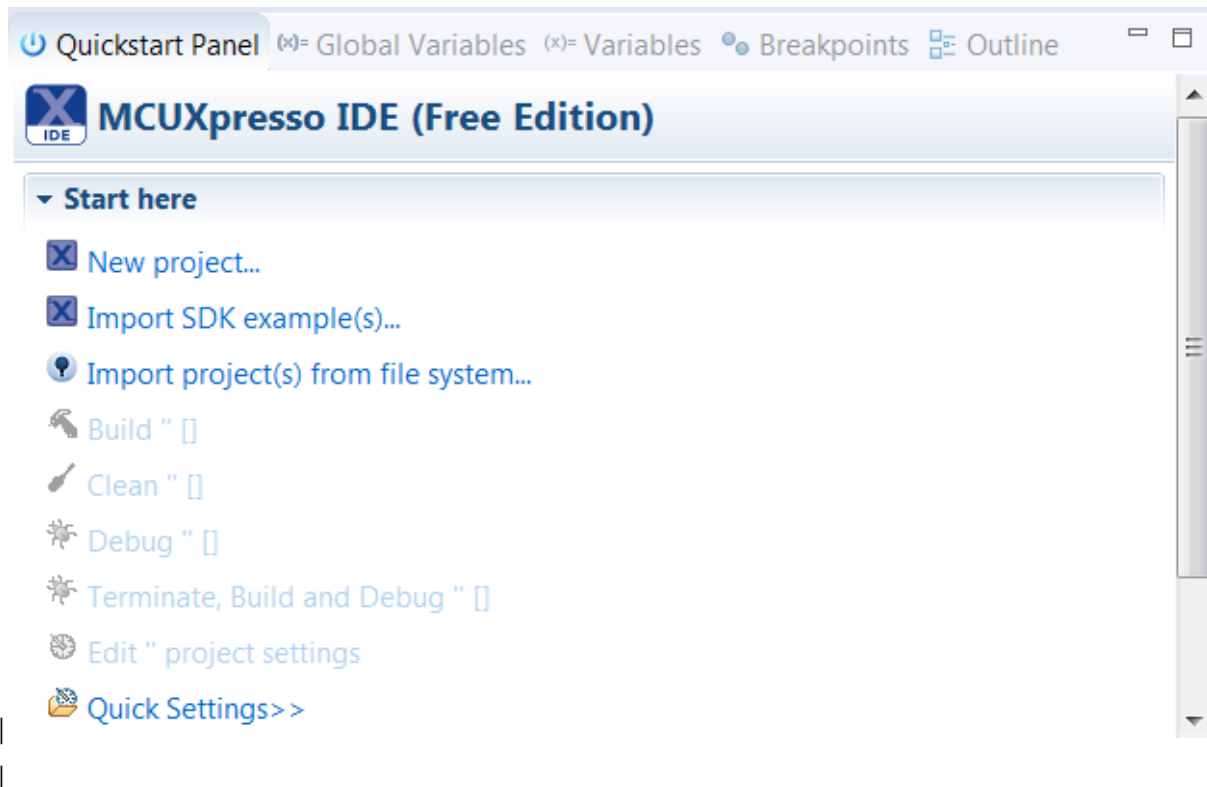
- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from www.nxp.com/mcuxpresso. Recommended for customers using IAR Embedded Workbench, Keil MDK µVision, or Arm GCC.
- **Online version** available on mcuxpresso.nxp.com. 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.

MCUXpresso IDE New Project Wizard

MCUXpresso IDE features a new project wizard. The wizard provides functionality for the user to create new projects from the installed SDKs (and from pre-installed part support). It offers user the flexibility to select and change multiple builds. The wizard also includes a library and provides source code options. The source code is organized as software components, categorized as drivers, utilities, and middleware.

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in *Figure 1*.



For more details and usage of new project wizard, see the *MCUXpresso_IDE_User_Guide.pdf* in the MCUXpresso IDE installation folder.

How to determine com port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform. All NXP boards ship with a factory programmed, on-board debug interface, whether it's based on OpenSDA or the legacy P&E Micro OSJTAG interface. To determine what your specific board ships with, see [Default debug interfaces](#).

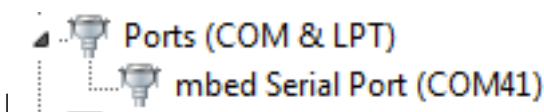
1. **Linux:** The serial port can be determined by running the following command after the USB Serial is connected to the host:

```
$ dmesg | grep "ttyUSB"
[503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
[503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
```

There are two ports, one is Cortex-A core debug console and the other is for Cortex M4.

2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click on the **Start** menu and type **Device Manager** in the search bar.
3. In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names will be different for all the NXP boards.

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



2. ****OpenSDA – P&E Micro****:

```

```

|

3. ****OpenSDA – J-Link****:

```

```

|

4. ****P&E Micro OSJTAG****:

```

```

|

5. ****MRB-KW01****:

```

```

|

How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to override the default IRQ handler. For example, to override the default PIT_IRQHandler define in startup_DEVICE.s, application code like app.c can be implement like:

```
c
void PIT_IRQHandler(void)
{
    // Your code
}
```

When application file is CPP file, like app.cpp, then `extern "C"` should be used to ensure the function prototype alignment.

```
cpp
extern "C" {
    void PIT_IRQHandler(void);
}
void PIT_IRQHandler(void)
{
    // Your code
}
```

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.

Note: The *OpenSDA details* column in *Table 1* is not applicable to LPC.

Hardware platform	Default interface	OpenSDA details
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
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
LPCXpresso55s36	CMSIS-DAP	N/A
LPCXpresso55s69	CMSIS-DAP	N/A
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0
MIMXRT1170-EVK	CMSIS-DAP	N/A
TWR-K21D50M	P&E Micro OSJTAG	N/AOpenSDA 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

continues on next page

Table 1 – continued from previous page

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

Updating debugger firmware

Updating LPCXpresso board firmware The LPCXpresso hardware platform comes with a CMSIS-DAP-compatible debug interface (known as LPC-Link2). This firmware in this debug interface may be updated using the host computer utility called LPCScript. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

Note: If MCUXpresso IDE is used and the jumper making DFULink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), LPC-Link2 debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the CMSIS-DAP firmware to the probe before flash memory programming (after clicking **Debug**). Using DFU mode ensures most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the LPCScript utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from www.nxp.com/lpcutilities.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScript user guide (www.nxp.com/lpcutilities, select **LPCScript**, and then the documentation tab).

1. Install the LPCScript utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labelled DFULink).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the LPCScript installation directory (<LPCScript install dir>).

1. To program CMSIS-DAP debug firmware: <LPCScript install dir>/scripts/program_CMSIS
2. To program J-Link debug firmware: <LPCScript install dir>/scripts/program_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Re-power the board by removing the USB cable and plugging it in again.

Parent topic: [Updating debugger firmware](#)

1.3 Getting Started with MCUXpresso SDK GitHub

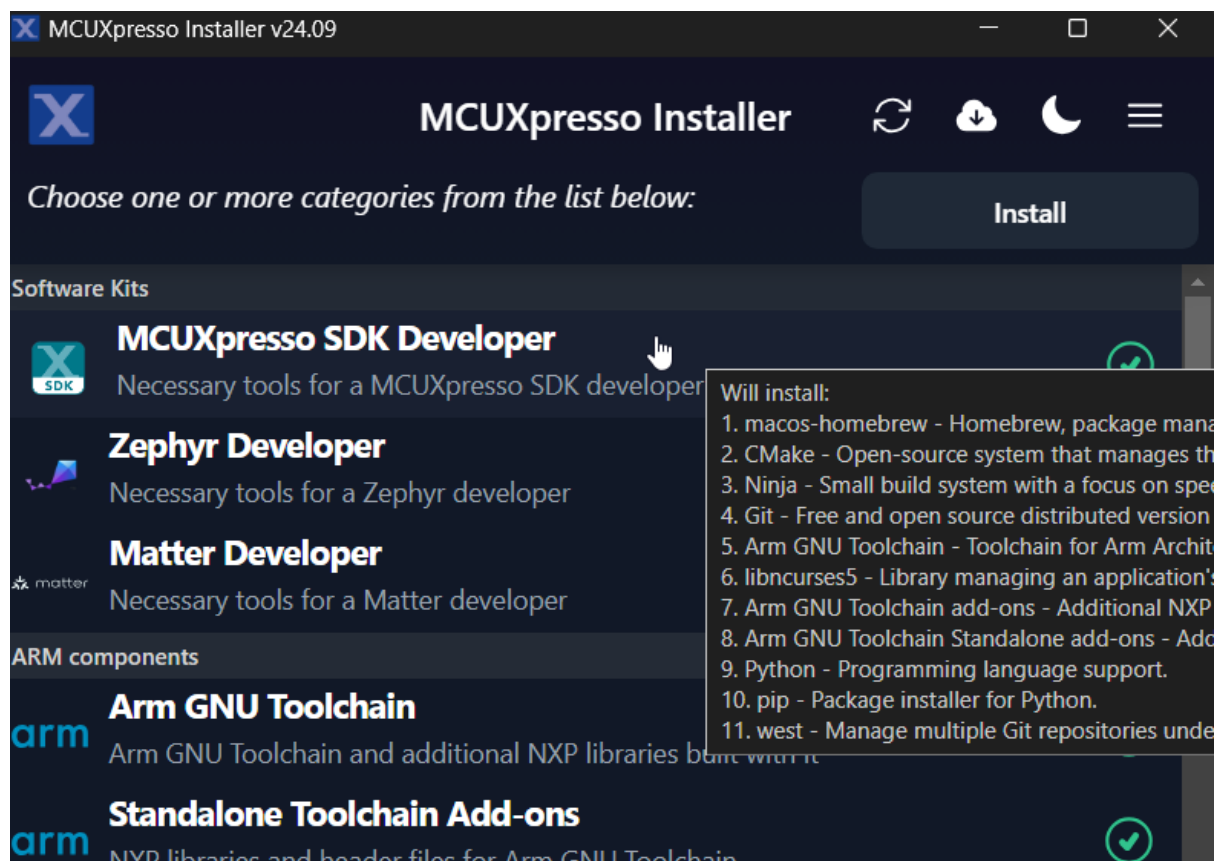
1.3.1 Getting Started with MCUXpresso SDK Repository

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.

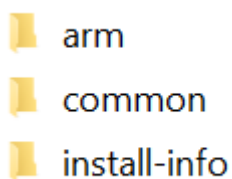
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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_DIR	c:\NXP\zephyr-sdk-<version> for windows/opt/zephyr-sdk-<version> for Linux	– toolchain zephyr
CodeWarrior	CW_DIR	C:\Freescall\CW MCU v11.2 for windowsCodeWarrior is not supported with Linux	– toolchain code-warrior
Xtensa	XCC_DIR	C:\xtensa\XtDevTools\install\tools\RI-2023.11-win32\XtensaTools for windows/opt/xtensa/XtDevTools/install/tools/RI-2023.11-Linux/XtensaTools for Linux	– toolchain xtensa
NXP S32Compiler RISC-V Zen-V	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

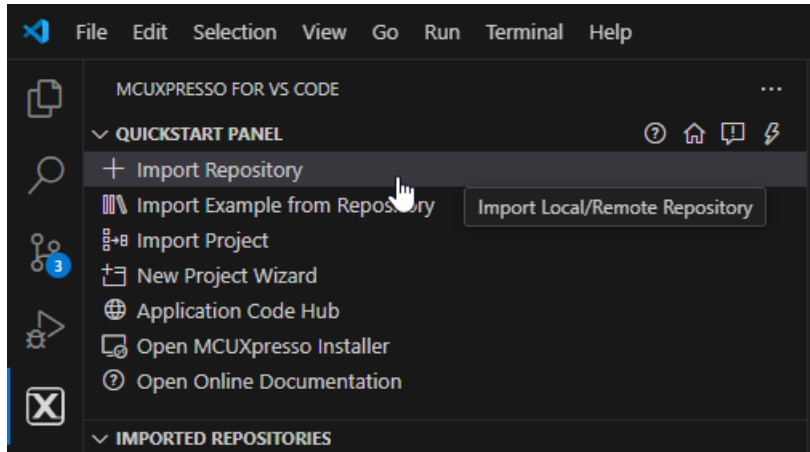
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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](#).

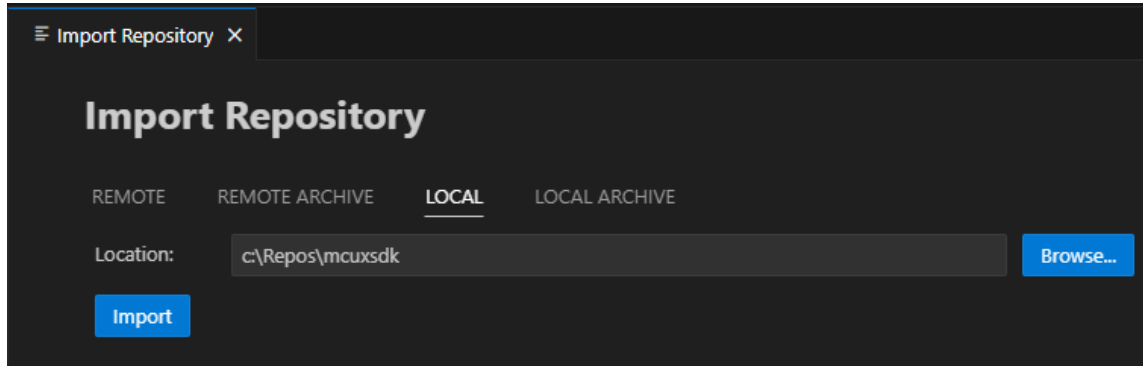
To build an example application:

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

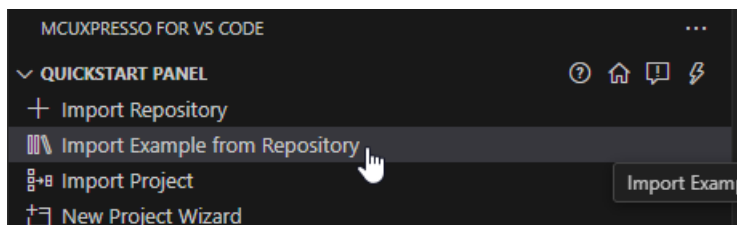


Note: You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

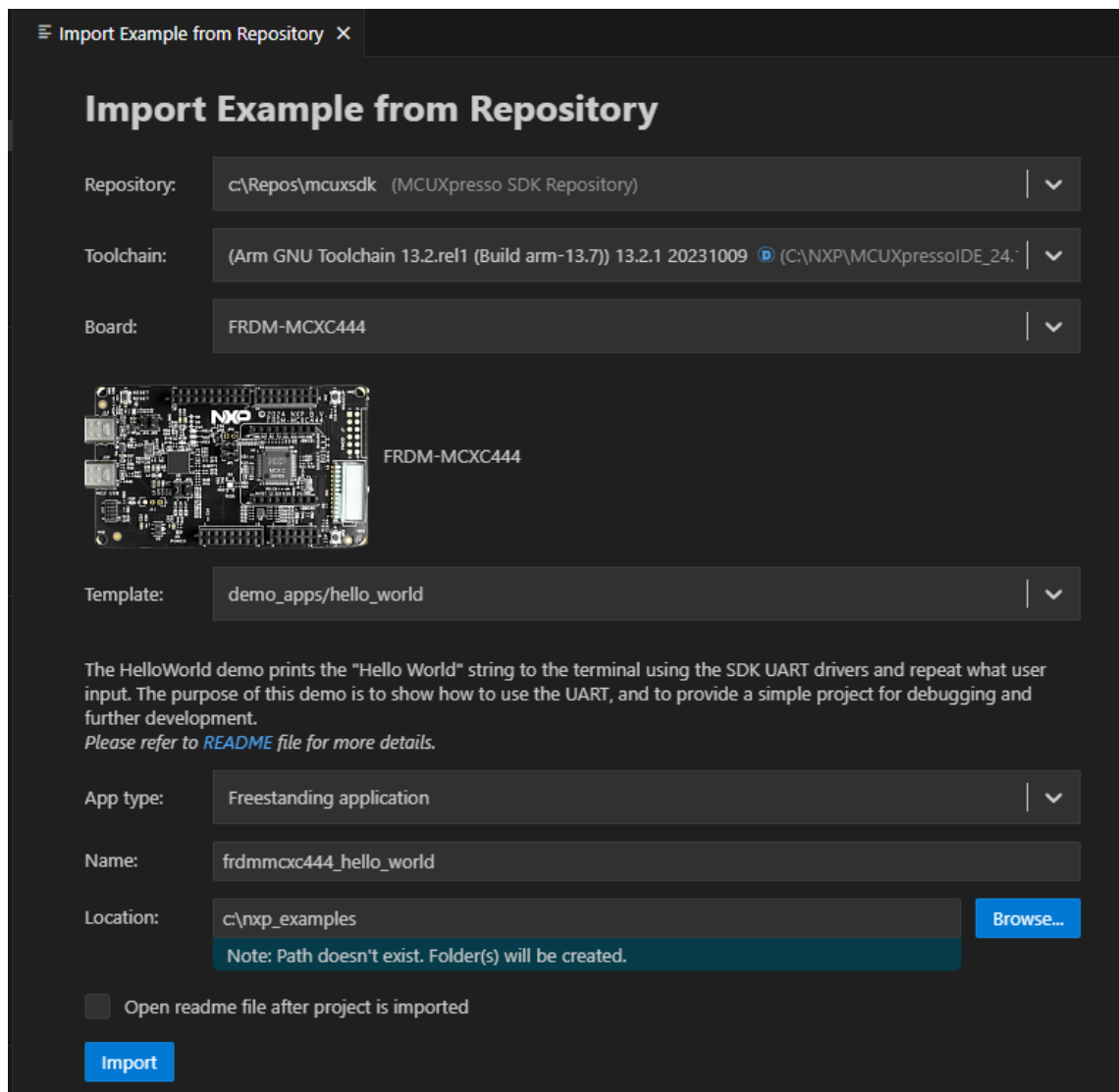
Select **Local** if you've already obtained the SDK as seen in [Get MCUXpresso SDK Repo](#). Select your location and click **Import**.



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

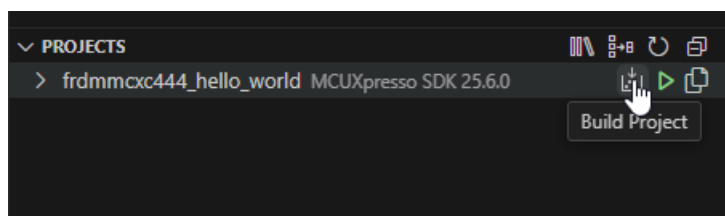


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.



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

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



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

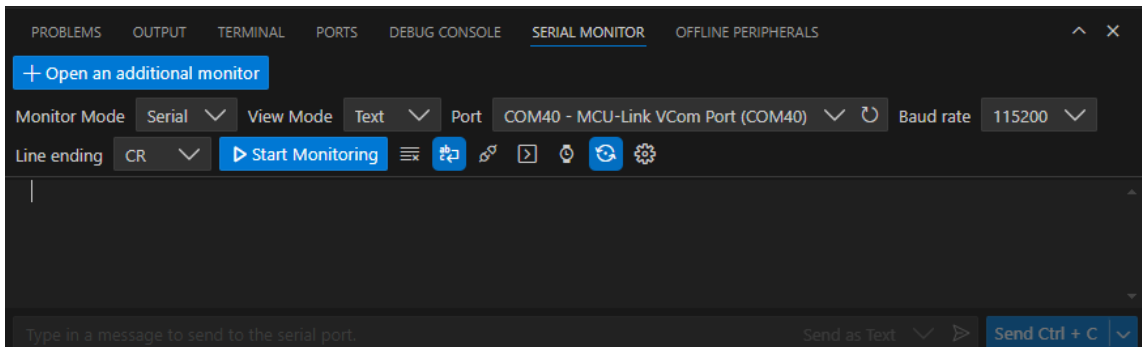
```

[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/MCX/MCX444/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.

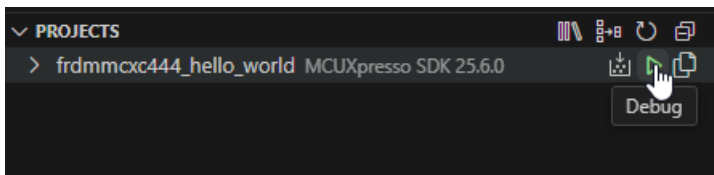
```

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

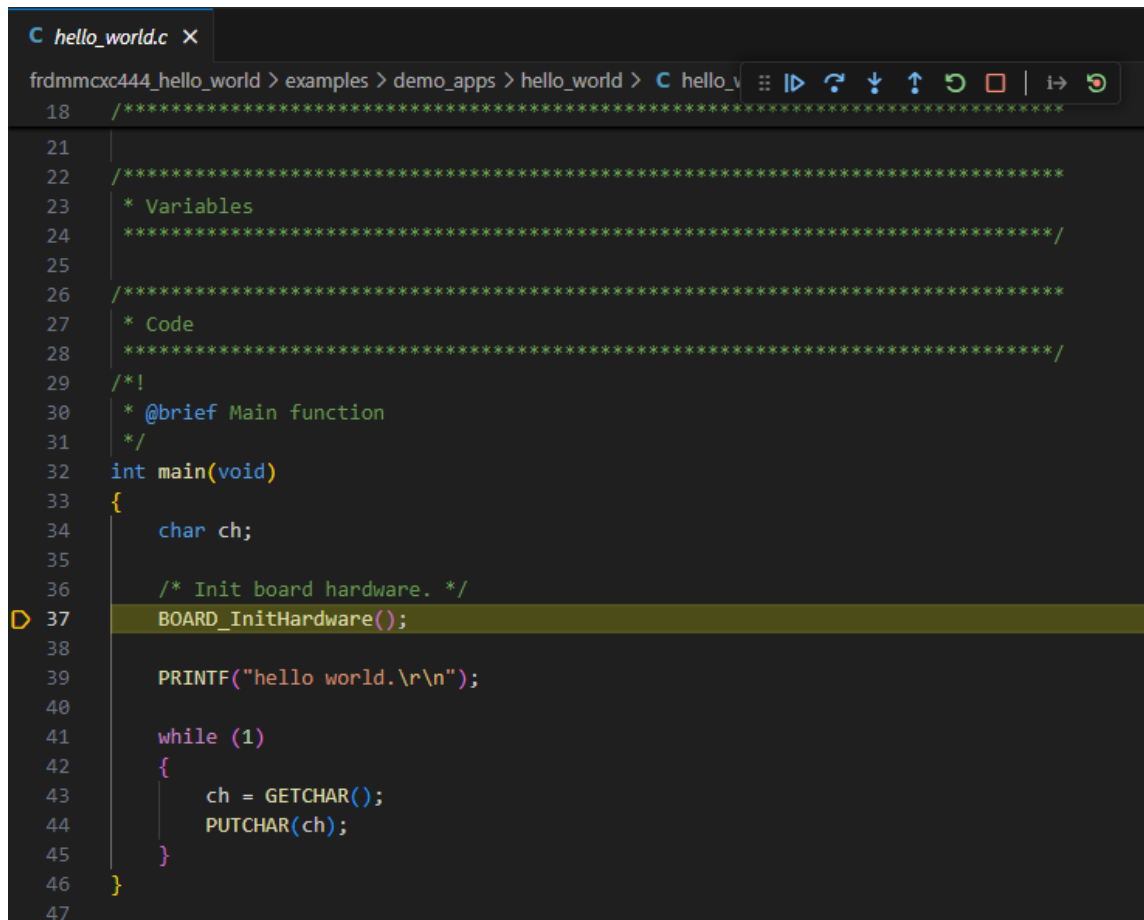
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



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.

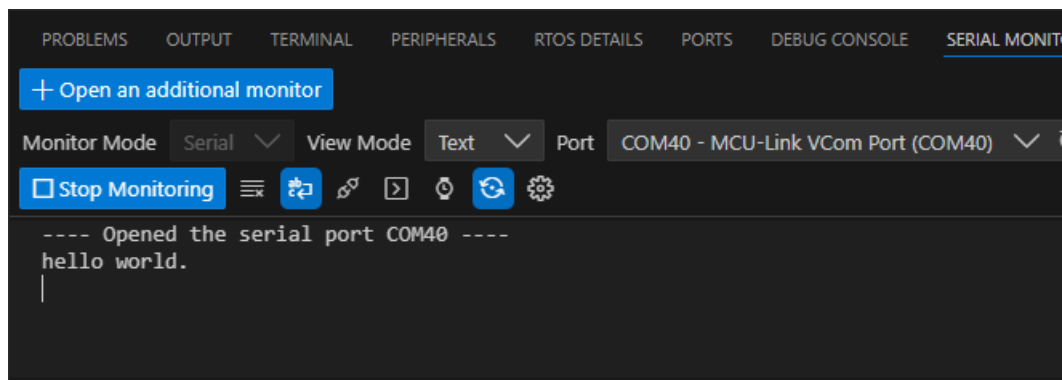


```

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)

```

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

```

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

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

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

Here are some typical usages for generating a SDK example:

```

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

For more details, please refer to System build.

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

1. Run cmake configuration

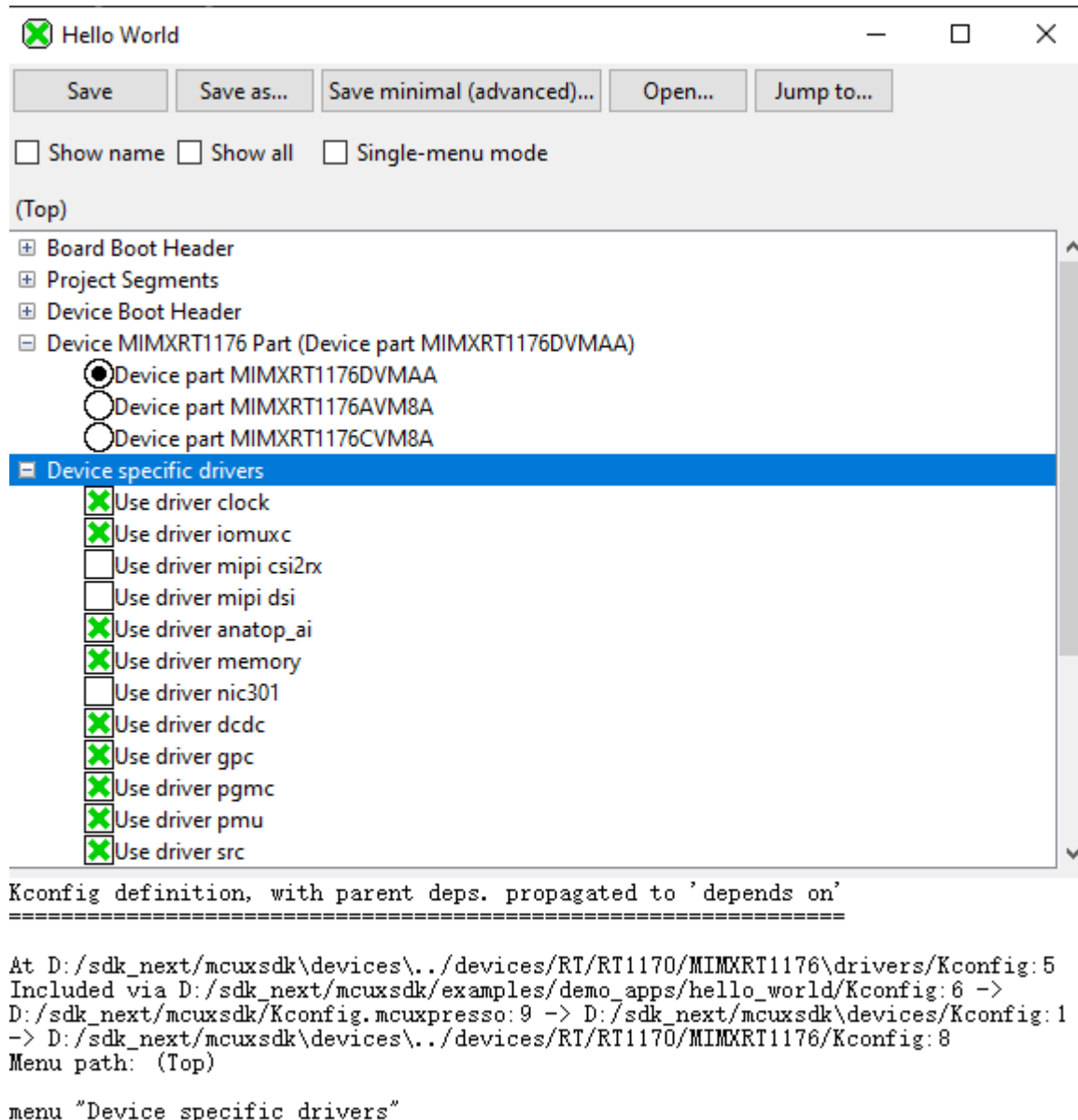
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like



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

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

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

Flash the hello_world example:

```
west flash -r linkserver
```

Debug Start a gdb interface by following command:

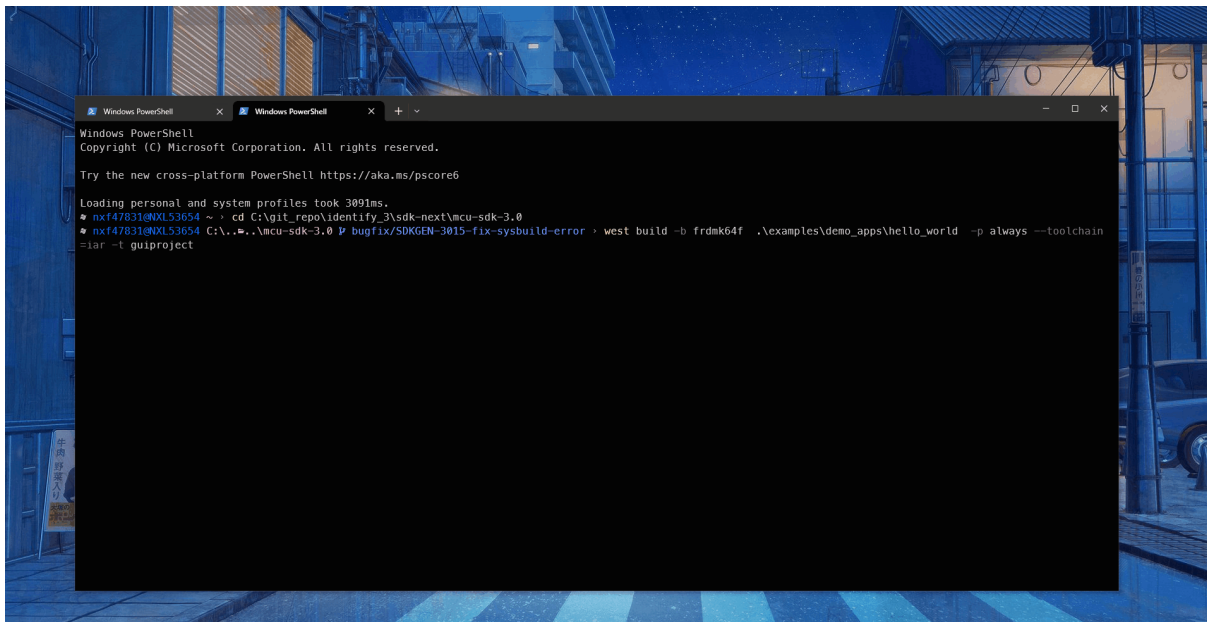
```
west debug -r linkserver
```

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

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

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

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



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

1.4 Release Notes

1.4.1 MCUXpresso SDK Release Notes

Overview

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

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

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

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

MCUXpresso SDK

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

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

Development tools

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

- 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			
LPCX- presso55S36	LPC55S36JBD100, LPC5534JHI48, LPC5536JBD100, LPC5536JBD64, LPC5536JHI48	LPC55S36JHI48,	LPC5534JBD100,	LPC5534JBD64,

MCUXpresso SDK release package

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

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

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

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

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Middleware

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

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

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

TF-M Trusted Firmware - M Library

PSA Test Suite Arm Platform Security Architecture Test Suite

Motor Control Software (ACIM, BLDC, PMSM) Motor control examples.

Multicore Multicore Software Development Kit

mbedTLS mbedtls SSL/TLS library v3.x

mbedTLS mbedtls SSL/TLS library v2.x

Metering Metering Library

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).

Audio Voice components Audio Voice components for MCU

Maestro Audio Framework for MCU Maestro Audio Framework library for MCU

LVGL LVGL Open Source Graphics Library

LittleFS LittleFS filesystem stack

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.

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

NXP ELS PKC ELS PKC crypto library

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.

LPCXpresso55S36 freertos_tzm examples

LPCXpresso55S36 freertos_tzm examples run fails on IAR v9.20.1 because of compiler issue. However, this issue is fixed on IAR v9.20.2.

LPCXpresso55S36 puf_v3 examples

LPCXpresso55S36 puf_v3 examples are not working in this release. This issue will be fixed in a future release.

MCUXpresso Config Tools v6.0.0 clone issue

- The lib file path cannot be converted when MDK examples are cloned.
- Not support to clone the TrustZone examples.

MCUXpresso IDE Rev. 11.5.x and GCC Arm Embedded 10.3-2021.x compatibility issue

- Cryptolib (CSS PKC) inline ASM is not compatible with new compiler syntax in MCUXpresso IDE, Rev. 11.5.x and GCC Am Embedded 10.3-2021.x.
- Support for new versions of mentioned compilers will be available in a future release.

TrustZone release example run fails in MCUXpresso IDE and ArmGCC 10-2020-q4-major

TrustZone release examples run fail in MCUXpresso IDE v11.4.1, this is IDE issue. This issue will be fixed in v11.5.0.

Examples `hello_world_ns`, `secure_faults_ns`, and `secure_faults_trdc_ns` have incorrect library path in GUI projects

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

Examples: `hello_world_ns`, `hello_world_s`, `secure_faults_ns`, `secure_faults_s`, `secure_faults_trdc_ns`, `secure_faults_trdc_s`

Affected toolchains: mdk, iar

Workaround: In the IDE project settings for the non-secure (`_ns`) project, find the linked library (named `hello_world_s_CMSE_lib.o`, or similar, depending on the example project) and replace the path to the library with `<build_directory>/<secure_world_project_folder>/<IDE>/`, replacing the subdirectory names with the build directory, the secure world project name, and IDE name.

Example `prince_rom` does not complete successfully after the Press any key to continue message

The `prince_rom` example hangs after prompting the user to press a button to continue program execution.

Examples: `prince_rom`

Affected toolchains: iar

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
-

ANACTRL

[2.4.0]

- Improvements
 - Added some interrupt flags for devices containing BOD1 and BOD2 interrupt controls.
 - Added a control macro to enable/disable the 32MHz Crystal oscillator code in current driver.
 - Added a feature macro for bit field ENA_96MHZCLK in FRO192M_CTRL.
 - Added a feature macro for bit field BODCORE_INT_ENABLE in BOD_DCDC_INT_CTRL.

[2.3.1]

- Bug Fixes
 - Added casts to prevent overflow caused by capturing large target clock.

[2.3.0]

- Improvements
 - Added AUX_BIAS control APIs.

[2.2.0]

- Improvements
 - Added some macros to separate the scenes that some bit fields are reserved for some devices.
 - Optimized the comments.
 - Optimized the code implementation inside some functions.

[2.1.2]

- Bug Fixes
 - Fixed MISRA C-2012 rule 10.3 and rule 17.7.

[2.1.1]

- Bug Fixes
 - Removed AnalogTestBus configuration to align with new header.

[2.1.0]

- Improvements
 - Updates for LPC55xx A1.
 - * Removed the control of bitfield FRO192M_CTRL_ENA_48MHZCLK, XO32M_CTRL_ACBUF_PASS_ENABLE.
 - * Removed status bits in ANACTRL_STATUS: PMU_ID OSC_ID FINAL_TEST_DONE_VECT.
 - * Removed API ANACTRL_EnableAdcVBATDivider() and APIs which operate the RingOSC registers.
 - * Removed the configurations of 32 MHz Crystal oscillator voltage source supply control register.
 - * Added API ANACTRL_ClearInterrupts().

[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.
-

CACHE64

[2.0.12]

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

[2.0.11]

- Bug Fixes
 - Fixed CERT INT30-C violations: check and guarantee address plus size is equal or smaller than UINT32_MAX.

[2.0.10]

- Improvements
 - Updated `CACHE64_InvalidateCacheByRange()`, `CACHE64_CleanCacheByRange()` and `CACHE64_CleanInvalidateCacheByRange()` to support some platforms that multiple regions in the memory map are remapped to create a continuous address space.

[2.0.9]

- Improvements
 - Removed `assert(false)` in `CACHE64_GetInstanceByAddr`.

[2.0.8]

- Improvements
 - Updated function `CACHE64_GetInstanceByAddr()` to support some devices that provide alias of cacheable memory section.

[2.0.7]

- Improvements
 - Check input parameter “size_byte” must be larger than 0.

[2.0.6]

- Bug Fixes
 - Fixed overflow for `CACHE64_GetInstanceByAddr()`/`CACHE64_CleanCacheByRange()`/`CACHE64_InvalidateCacheByRange()` APIs.

[2.0.5]

- Improvement
 - Made use of `FSL_FEATURE_CACHE64_CTRL_HAS_NO_WRITE_BUF` feature

[2.0.4]

- Improvement
 - Disable cache policy feature on SoC without `CACHE64_POLSEL` IP.
- Bug Fixes
 - Fixed doxygen issue.

[2.0.3]

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

[2.0.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4 and 14.4.
 - Fixed doxygen issue.

[2.0.1]

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

[2.0.0]

- Initial version.
-

CACHE LPCAC

[2.2.1]

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

[2.2.0]

- Improvements
 - Used new add macros FSL_FEATURE_SYSCON_HAS_LPCAC_CTRL_PARITY_MISS_EN_BIT and FSL_FEATURE_SYSCON_HAS_LPCAC_CTRL_PARITY_FAULT_EN_BIT to support some platforms which PARITY_MISS_EN and PARITY_FAULT_EN bit may be reserved.

[2.1.1]

- Bug Fixes
 - Fixed an issue of L1CACHE_InvalidateCodeCache() function, to clean cache the LPCAC_CTRL[CLR_LPCAC] should be set not clear.

[2.1.0]

- Improvements
 - Supported more features, such as write buffer contron, write buffer limit and so on.

[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

[2.3.3]

- Bug Fixes
 - Fixed an issue that ss_progmodfm_t, ss_progmoddp_t, and ss_modwvctrl_t use wrong shift value.

[2.3.2]

- Improvements
 - Add errata workaround for pll lock bit in CLOCK_SetPLL0Freq() and CLOCK_SetPLL1Freq().

[2.0.0]

- Initial version.
-

CMP

[2.2.1]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.1, rule 10.4, and rule 17.7.

[2.2.0]

- Improvements:
 - Added API to configure the sampling mode and clock divider of the CMP Filter.
 - Supported CMP filter sampling mode configuration.

[2.1.0]

- New Features:
 - Added API to get default CMP user configuration structure.
 - Supported CMP filter clock divider settings.
 - Combined the settings of VREF source and VREF value into one API `CMP_SetVREF()`.
 - Extracted CMP input source selection from `CMP_Init()` to `CMP_SetInputChannels()`.
- Improvements:
 - Formatted API naming, variable naming and comment style for better readability.
 - Added comments for APIs in source file.

[2.0.1]

- Bug Fixes
 - Fixed missing 'const' qualifier for structure variable in function parameter.

[2.0.0]

- Initial version.
-

COMMON

[2.6.1]

- Improvements
 - Support Cortex M23.

[2.6.0]

- Bug Fixes
 - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

- Improvements
 - Enable irq's 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.
-

CRC

[2.0.4]

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

[2.0.3]

- Bug fix:
 - Fix MISRA issues.

[2.0.2]

- Bug fix:
 - Fix MISRA issues.

[2.0.1]

- Bug fix:
 - DATA and DATALL macro definition moved from header file to source file.

[2.0.0]

- Initial version.
-

CTIMER

[2.3.3]

- Bug Fixes
 - Fix CERT INT30-C INT31-C issue.
 - Make API CTIMER_SetupPwm and CTIMER_UpdatePwmDutycycle return fail if pulse width register overflow.

[2.3.2]

- Bug Fixes
 - Clear unexpected DMA request generated by RESET_PeripheralReset in API CTIMER_Init to avoid trigger DMA by mistake.

[2.3.1]

- Bug Fixes
 - MISRA C-2012 issue fixed: rule 10.7 and 12.2.

[2.3.0]

- Improvements
 - Added the CTIMER_SetPrescale(), CTIMER_GetCaptureValue(), CTIMER_EnableResetMatchChannel(), CTIMER_EnableStopMatchChannel(), CTIMER_EnableRisingEdgeCapture(), CTIMER_EnableFallingEdgeCapture(), CTIMER_SetShadowValue(), APIs Interface to reduce code complexity.

[2.2.2]

- Bug Fixes
 - Fixed SetupPwm() API only can use match 3 as period channel issue.

[2.2.1]

- Bug Fixes
 - Fixed use specified channel to setting the PWM period in SetupPwmPeriod() API.
 - Fixed Coverity Out-of-bounds issue.

[2.2.0]

- Improvements
 - Updated three API Interface to support Users to flexibly configure the PWM period and PWM output.
- Bug Fixes
 - MISRA C-2012 issue fixed: rule 8.4.

[2.1.0]

- Improvements
 - Added the CTIMER_GetOutputMatchStatus() API Interface.
 - Added feature macro for FSL_FEATURE_CTIMER_HAS_NO_CCR_CAP2 and FSL_FEATURE_CTIMER_HAS_NO_IR_CR2INT.

[2.0.3]

- Bug Fixes
 - MISRA C-2012 issue fixed: rule 10.3, 10.4, 10.6, 10.7 and 11.9.

[2.0.2]

- New Features
 - Added new API “CTIMER_GetTimerCountValue” to get the current timer count value.
 - Added a control macro to enable/disable the RESET and CLOCK code in current driver.
 - Added a new feature macro to update the API of CTimer driver for lpc8n04.

[2.0.1]

- Improvements
 - API Interface Change
 - * Changed API interface by adding CTIMER_SetupPwmPeriod API and CTIMER_UpdatePwmPulsePeriod API, which both can set up the right PWM with high resolution.

[2.0.0]

- Initial version.
-

DAC

[2.1.3]

- Improvements
 - ‘dac_config_t’ already contains a member ‘enableOpampBuffer’, but it is not used. This update uses ‘enableOpampBuffer’ to control the configuration of GCR[BUF_EN].

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- New Features
 - Added support for period trigger mode.
 - Added support for sync trigger between dac instances.
 - Added support for configuring DAC sync cycles.
 - Added support for internal reference current selection.
 - Enabled buffer mode manually for K4 series board.

[2.0.2]

- Bug Fixes
 - Fixed MISRA C-2012 rule 10.3 and rule 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.
-

LPC_DMA**[2.5.3]**

- Improvements
 - Add assert in DMA_SetChannelXferConfig to prevent XFERCOUNT value overflow.

[2.5.2]

- Bug Fixes
 - Use separate “SET” and “CLR” registers to modify shared registers for all channels, in case of thread-safe issue.

[2.5.1]

- Bug Fixes
 - Fixed violation of the MISRA C-2012 rule 11.6.

[2.5.0]

- Improvements
 - Added a new api DMA_SetChannelXferConfig to set DMA xfer config.

[2.4.4]

- Bug Fixes
 - Fixed the issue that DMA_IRQHandle might generate redundant callbacks.
 - Fixed the issue that DMA driver cannot support channel bigger then 32.
 - Fixed violation of the MISRA C-2012 rule 13.5.

[2.4.3]

- Improvements
 - Added features FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZEn/FSL_FEATURE_DMA0_DESCRIPTOR_ALIGN_SIZEn to support the descriptor align size not constant in the two instances.

[2.4.2]

- Bug Fixes
 - Fixed violation of the MISRA C-2012 rule 8.4.

[2.4.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 5.7, 8.3.

[2.4.0]

- Improvements
 - Added new APIs DMA_LoadChannelDescriptor/DMA_ChannelIsBusy to support polling transfer case.
- Bug Fixes
 - Added address alignment check for descriptor source and destination address.
 - Added DMA_ALLOCATE_DATA_TRANSFER_BUFFER for application buffer allocation.
 - Fixed the sign-compare warning.
 - Fixed violations of the MISRA C-2012 rules 18.1, 10.4, 11.6, 10.7, 14.4, 16.3, 20.7, 10.8, 16.1, 17.7, 10.3, 3.1, 18.1.

[2.3.0]

- Bug Fixes
 - Removed DMA_HandleIRQ prototype definition from header file.
 - Added DMA_IRQHandle prototype definition in header file.

[2.2.5]

- Improvements
 - Added new API DMA_SetupChannelDescriptor to support configuring wrap descriptor.
 - Added wrap support in function DMA_SubmitChannelTransfer.

[2.2.4]

- Bug Fixes
 - Fixed the issue that macro DMA_CHANNEL_CFER used wrong parameter to calculate DSTINC.

[2.2.3]

- Bug Fixes
 - Improved DMA driver Deinit function for correct logic order.
- Improvements
 - Added API DMA_SubmitChannelTransferParameter to support creating head descriptor directly.

- Added API DMA_SubmitChannelDescriptor to support ping pong transfer.
- Added macro DMA_ALLOCATE_HEAD_DESCRIPTOR/DMA_ALLOCATE_LINK_DESCRIPTOR to simplify DMA descriptor allocation.

[2.2.2]

- Bug Fixes
 - Do not use software trigger when hardware trigger is enabled.

[2.2.1]

- Bug Fixes
 - Fixed Coverity issue.

[2.2.0]

- Improvements
 - Changed API DMA_SetupDMADescriptor to non-static.
 - Marked APIs below as deprecated.
 - * DMA_PrepareTransfer.
 - * DMA_Submit transfer.
 - Added new APIs as below:
 - * DMA_SetChannelConfig.
 - * DMA_PrepareChannelTransfer.
 - * DMA_InstallDescriptorMemory.
 - * DMA_SubmitChannelTransfer.
 - * DMA_SetChannelConfigValid.
 - * DMA_DoChannelSoftwareTrigger.
 - * DMA_LoadChannelTransferConfig.

[2.0.1]

- Improvements
 - Added volatile for DMA descriptor member xfercfg to avoid optimization.

[2.0.0]

- Initial version.
-

DMIC

[2.3.3]

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

[2.3.2]

- New Features
 - Supported 4 channels in driver.

[2.3.1]

- Bug Fixes
 - Fixed the issue that DMIC_EnableChannelDma and DMIC_EnableChannelFifo did not clean relevant bits.

[2.3.0]

- Improvements
 - Added new apis DMIC_ResetChannelDecimator/DMIC_EnableChannelGlobalSync/DMIC_DisableChannelGlobalSync

[2.2.1]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 14.4, 17.7, 10.4, 10.3, 10.8, 14.3.

[2.2.0]

- Bug Fixes
 - Corrected the usage of feature FSL_FEATURE_DMIC_IO_HAS_NO_BYPASS.

[2.1.1]

- Improvements
 - Added feature FSL_FEATURE_DMIC_HAS_NO_IOCFCG for IOCFCG register.

[2.1.0]

- New Features
 - Added API DMIC_EnableChannelInterrupt/DMIC_EnableChannelDma to replace API DMIC_SetOperationMode.
 - Added API DMIC_SetIOCFCG and marked DMIC_ConfigIO as deprecated.
 - Added API DMIC_EnableChannelSignExtend to support sign extend feature.

[2.0.5]

- Improvements
 - Changed some parameters' value of DMIC_FifoChannel API, such as enable, resetn, and trig_level. This is not possible for the current code logic, so it improves the DMIC_FifoChannel logic and fixes incorrect math logic.

[2.0.4]

- Bug Fixes
 - Fixed the issue that DMIC DMA driver(ver2.0.3) did not support calling DMIC_TransferReceiveDMA in DMA callback as it did before version 2.0.3. But calling DMIC_TransferReceiveDMA in callback is not recommended.

[2.0.3]

- New Features
- Supported linked transfer in DMIC DMA driver.
- Added new API DMIC_EnableChannelFifo/DMIC_DoFifoReset/DMIC_InstallDMADescriptor.

[2.0.2]

- New Features
 - Supported more channels in driver.

[2.0.1]

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

[2.0.0]

- Initial version.
-

DMIC_DMA**[2.4.2]**

- Bug Fixes
 - Fixed coverity High Impact finding

[2.4.1]

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

[2.4.0]

- Bug Fixes
 - Fixed the issue that DMIC_TransferAbortReceiveDMA can not disable dmic and dma request issue.

[2.3.1]

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

[2.3.0]

- Refer DMIC driver change log 2.0.1 to 2.3.0
-

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.
-

FLEXCOMM

[2.0.2]

- Bug Fixes
 - Fixed typos in FLEXCOMM15_DriverIRQHandler().
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.
- Improvements
 - Added instance calculation in FLEXCOMM16_DriverIRQHandler() to align with Flex-comm 14 and 15.

[2.0.1]

- Improvements
 - Added more IRQHandler code in drivers to adapt new devices.

[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_CONBINATION.

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

FLEXSPI DMA Driver**[2.2.1]**

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

[2.2.0]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3.
- New Features
 - Updated name of FLEXSPI_TransferGetTransferCountDMA API.

[2.1.1]

- New Features
 - Updated driver to support feature FSL_FEATURE_FLEXSPI_DMA_MULTIPLE_DES.

[2.1.0]

- Bug Fixes
 - Updated enumeration flexspi_dma_transfer_nsize_t and remove the unsupported items.
- New Features
 - Updated driver for deprecating the multiple linked descriptors inside FLEXSPI_TransferDMA, only up to one linked descriptor is needed according to hardware update.

[2.0.0]

- Initial version.
-

FREQME

[2.1.2]

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

[2.1.1]

- Fixed MISRA issues.

[2.1.0]

- Updated register name.

[2.0.0]

- Initial version.
-

GINT

[2.1.1]

- Improvements
 - Added support for platforms with PORT_POL and PORT_ENA registers without arrays.

[2.1.0]

- Improvements
 - Updated for platforms which only has one port.

[2.0.3]

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

[2.0.2]

- Bug Fixes
 - Fixed issue for MISRA-2012 check.
 - * Fixed rule 17.7.

[2.0.1]

- Added control macro to enable/disable the RESET and CLOCK code in current driver.

[2.0.0]

- Initial version.
-

GPIO**[2.1.7]**

- Improvements
 - Enhanced GPIO_PinInit to enable clock internally.

[2.1.6]

- Bug Fixes
 - Clear bit before set it within GPIO_SetPinInterruptConfig() API.

[2.1.5]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 3.1, 10.6, 10.7, 17.7.

[2.1.4]

- Improvements
 - Added API GPIO_PortGetInterruptStatus to retrieve interrupt status for whole port.
 - Corrected typos in header file.

[2.1.3]

- Improvements
 - Updated “GPIO_PinInit” API. If it has DIRCLR and DIRSET registers, use them at set 1 or clean 0.

[2.1.2]

- Improvements
 - Removed deprecated APIs.

[2.1.1]

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

[2.1.0]

- New Features
 - Added GPIO initialize API.

[2.0.0]

- Initial version.
-

HSCMP

[2.0.4]

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

[2.0.3]

- Bug Fixes:
 - Fixed the violation of MISRA-2012 rule 17.7.

[2.0.2]

- Bug Fixes:
 - The current API `LPCMP_ClearStatusFlags` has to check w1c bits.

[2.0.1]

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

[2.0.0]

- Initial version.
-

I2C

[2.3.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1.
 - Fixed issue that if master only sends address without data during I2C interrupt transfer, address nack cannot be detected.

[2.3.2]

- Improvement
 - Enable or disable timeout option according to enableTimeout.
- Bug Fixes
 - Fixed timeout value calculation error.
 - Fixed bug that the interrupt transfer cannot recover from the timeout error.

[2.3.1]

- Improvement
 - Before master transfer with transactional APIs, enable master function while disable slave function and vice versa for slave transfer to avoid the one affecting the other.
- Bug Fixes
 - Fixed bug in I2C_SlaveEnable that the slave enable/disable should not affect the other register bits.

[2.3.0]

- Improvement
 - Added new return codes kStatus_I2C_EventTimeout and kStatus_I2C_SclLowTimeout, and added the check for event timeout and SCL timeout in I2C master transfer.
 - Fixed bug in slave transfer that the address match event should be invoked before not after slave transmit/receive event.

[2.2.0]

- New Features
 - Added enumeration `_i2c_status_flags` to include all previous master and slave status flags, and added missing status flags.
 - Modified `I2C_GetStatusFlags` to get all I2C flags.
 - Added API `I2C_ClearStatusFlags` to clear all clearable flags not just master flags.
 - Modified master transactional APIs to enable bus event timeout interrupt during transfer, to avoid glitch on bus causing transfer hangs indefinitely.
- Bug Fixes
 - Fixed bug that status flags and interrupt enable masks share the same enumerations by adding enumeration `_i2c_interrupt_enable` for all master and slave interrupt sources.

[2.1.0]

- Bug Fixes
 - Fixed bug that during master transfer, when master is nacked during slave probing or sending subaddress, the return status should be `kStatus_I2C_Addr_Nak` rather than `kStatus_I2C_Nak`.
- Bug Fixes
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.4, 13.5.
- New Features
 - Added macro `I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK`, so that user can configure whether to ignore the last byte being nacked by slave during master transfer.

[2.0.8]

- Bug Fixes
 - Fixed `I2C_MasterSetBaudRate` issue that `MSTSCLLOW` and `MSTSCLHIGH` are incorrect when `MSTTIME` is odd.

[2.0.7]

- Bug Fixes
 - Two dividers, `CLKDIV` and `MSTTIME` are used to configure baudrate. According to reference manual, in order to generate 400kHz baudrate, the clock frequency after `CLKDIV` must be less than 2mHz. Fixed the bug that, the clock frequency after `CLKDIV` may be larger than 2mHz using the previous calculation method.
 - Fixed MISRA 10.1 issues.
 - Fixed wrong baudrate calculation when feature `FSL_FEATURE_I2C_PREPCLKFRG_8MHZ` is enabled.

[2.0.6]

- New Features
 - Added master timeout self-recovery support for feature `FSL_FEATURE_I2C_TIMEOUT_RECOVERY`.
- Bug Fixes
 - Eliminated IAR Pa082 warning.
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.

[2.0.5]

- Bug Fixes
 - Fixed wrong assignment for `datasize` in `I2C_InitTransferStateMachineDMA`.
 - Fixed wrong working flow in `I2C_RunTransferStateMachineDMA` to ensure master can work in no start flag and no stop flag mode.

- Fixed wrong working flow in I2C_RunTransferStateMachine and added kReceive-DataBeginState in _i2c_transfer_states to ensure master can work in no start flag and no stop flag mode.
- Fixed wrong handle state in I2C_MasterTransferDMAHandleIRQ. After all the data has been transfered or nak is returned, handle state should be changed to idle.
- Improvements
 - Rounded up the calculated divider value in I2C_MasterSetBaudRate.

[2.0.4]

- Improvements
 - Updated the I2C_WATI_TIMEOUT macro to unified name I2C_RETRY_TIMES
 - Updated the “I2C_MasterSetBaudRate” API to support baudrate configuration for feature QN9090.
- Bug Fixes
 - Fixed build warnning caused by uninitialized variable.
 - Fixed COVERITY issue of unchecked return value in I2C_RTOS_Transfer.

[2.0.3]

- Improvements
 - Unified the component full name to FLEXCOMM I2C(DMA/FREERTOS) driver.

[2.0.2]

- Improvements
 - In slave IRQ:
 1. Changed slave receive process to first set the I2C_SLVCTL_SLVCONTINUE_MASK to acknowledge the received data, then do data receive.
 2. Improved slave transmit process to set the I2C_SLVCTL_SLVCONTINUE_MASK immediately after writing the data.

[2.0.1]

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

[2.0.0]

- Initial version.
-

I2S

[2.3.2]

- Bug Fixes
 - Fixed warning for comparison between pointer and integer.

[2.3.1]

- Bug Fixes
 - Updated the value of TX/RX software transfer state machine after transfer contents are submitted to avoid race condition.

[2.3.0]

- Improvements
 - Added api I2S_InstallDMADescriptorMemory/I2S_TransferSendLoopDMA/I2S_TransferReceiveLoopDMA to support loop transfer.
 - Added api I2S_EmptyTxFifo to support blocking flush tx fifo.
 - Updated api I2S_TransferAbortDMA by removed the blocking flush tx fifo from this function.
- Bug Fixes
 - Removed the while loop in abort transfer function to fix the dead loop issue under specific user case.

[2.2.2]

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

[2.2.1]

- Improvements
 - Added feature FSL_FEATURE_FLEXCOMM_INSTANCE_I2S_SUPPORT_SECONDARY_CHANNELn for the SOC has parts of instance support secondary channel.
- Bug Fixes
 - Added volatile statement for the state variable of i2s_handle and enable the mainline channel pair before enable interrupt to avoid the issue of code excution reordering which may cause the interrupt generated unexpectedly.

[2.2.0]

- Improvements
 - Added 8/16/24 bits mono data format transfer support in I2S driver.
 - Added new apis I2S_SetBitClockRate.
- Bug Fixes
 - Fixed the PA082 build warning.
 - Fixed the sign-compare warning.
 - Fixed violations of the MISRA C-2012 rules 10.4, 10.8, 11.9, 10.1, 11.3, 13.5, 11.8, 10.3, 10.7.
 - Fixed the Operand don't affect result Coverity issue.

[2.1.0]

- Improvements
 - Added a feature for the FLEXCOMM which supports I2S and has interconnection with DMIC.
 - Used a feature to control PDMDATA instead of I2S_CFG1_PDMDATA.
 - Added member bytesPerFrame in i2s_dma_handle_t, used for DMA transfer width configure, instead of using sizeof(uint32_t) hardcoded.
 - Used the macro provided by DMA driver to define the I2S DMA descriptor.
- Bug Fixes
 - Fixed the issue that I2S DMA driver always generated duplicate callback.

[2.0.3]

- New Features
 - Added a feature to remove configuration for the second channel on LPC51U68.

[2.0.2]

- New Features
 - Added ENABLE_IRQ handle after register I2S interrupt handle.

[2.0.1]

- Improvements
 - Unified the component full name to FLEXCOMM I2S (DMA) driver.

[2.0.0]

- Initial version.
-

I2S_DMA**[2.3.3]**

- Bug Fixes
 - Fixed data size limit does not match the macro DMA_MAX_TRANSFER_BYTES issue.

[2.3.2]

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

[2.3.1]

- Refer I2S driver change log 2.0.1 to 2.3.1
-

I3C

[2.14.3]

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

[2.14.2]

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

[2.14.1]

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

[2.14.0]

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

[2.13.1]

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

[2.13.0]

- New features
 - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
 - Set read termination with START at the same time in case unknown issue.
 - Set MCTRL[TYPE] as 0 for DDR force exit.
- Improvements
 - Added the API to reset device count assigned by ENTDAAs.

- Provided the method to set global macro I3C_MAX_DEVCNT to determine how many device addresses ENTDAAs can allocate at one time.
- Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

[2.12.0]

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

[2.11.1]

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

[2.11.0]

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

[2.10.8]

- Improvements
 - Support more instances.

[2.10.7]

- Improvements
 - Fixed the potential compile warning.

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- New features
 - Added I3C extended IBI data support.

[2.9.0]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.4]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

- Bug Fixes
 - Fixed `ki3C_SlaveMatchedFlag` interrupt is not properly handled in `I3C_SlaveTransferHandleIRQ` when it comes together with interrupt `ki3C_SlaveBusStartFlag`.
 - Fixed the inaccurate I2C baudrate calculation in `I3C_MasterSetBaudRate`.
 - Added new API `I3C_MasterGetIBIRules` to get registered IBI rules.
 - Added new variable `isReadTerm` in `struct_i3c_master_handle` for transfer state routine to check if `MCTRL.RDTERM` is configured for read transfer.
 - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
 - Fixed the slave `maxWriteLength` and `maxReadLength` does not be configured into `SMAXLIMITS` register issue.
 - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
 - Added `isHotJoin` in `i3c_slave_config_t` to request hot-join event during slave init.

[2.3.2]

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

[2.3.1]

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

[2.3.0]

- Improvements
 - Added I3C common driver APIs to initialize I3C with both master and slave configuration.

- Updated I3C master transfer callback to function set structure to include callback invoke for IBI event and slave2master event.
- Updated I3C master non-blocking transfer model and always enable the interrupts to be able to re-act to the slave start event and handle slave IBI.

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

I3C_DMA

[2.1.9]

- Improvements
 - Fixed Coverity CERT-C violations.

[2.1.8]

- Bug Fixes
 - Updated the logic to handle Rx termination and complete event to adapt different situation.
- Improvements
 - Added the MCTRLDONE flag check after STOP request to ensure the completion of whole transfer operation.

[2.1.7]

- Bug Fixes
 - Fixed the issue to use subaddress to read/write data with RT500/600 DMA.

[2.1.6]

- Improvements
 - Added the FSL_FEATURE_I3C_HAS_NO_MASTER_DMA_WDATA_REG to select the correct register to write data based on specific Soc.

[2.1.5]

- New features
 - Supported I3C HDR-DDR transfer with DMA.
- Improvements
 - Added workaround for RT500/600 I3C DMA transfer.
 - Removed I3C IRQ handler calling in the Tx EDMA callback. Previously driver doesn't use the END byte which can trigger the complete interrupt for controller sending and receiving, now let I3C event handler deal with I3C events.
 - Used linked DMA to transfer all I3C subaddress and data without handling of intermediate states, simplifying code logic.
 - Prepare the Tx DMA before I3C START to ensure there's no time delay between START and transmitting data.

[2.1.4]

- Improvements
 - Used linked DMA transfer to reduce the latency between DMA transfers previous data and the END byte.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
 - Fixed MISRA issue rule 9.1.

[2.1.0]

- Improvements
 - Deleted legacy IBI data request code.

[2.0.1]

- Bug Fixes
 - Fixed issue that bus STOP occurs when Tx FIFO still takes data.
- Improvements
 - Fixed the build warning due to armgcc strict check.

[2.0.0]

- Initial version.
-

INPUTMUX**[2.0.10]**

- Bug Fixes
 - Fixed CERT-C violations.

[2.0.9]

- Improvements
 - Use INPUTMUX_CLOCKS to initialize the inputmux module clock to adapt to multiple inputmux instances.
 - Modify the API base type from INPUTMUX_Type to void.

[2.0.8]

- Improvements
 - Updated a feature macro usage for function INPUTMUX_EnableSignal.

[2.0.7]

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

[2.0.6]

- Bug Fixes
 - Fixed the documentation wrong in API INPUTMUX_AttachSignal.

[2.0.5]

- Bug Fixes
 - Fixed build error because some devices has no sct.

[2.0.4]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rule 10.4, 12.2 in INPUTMUX_EnableSignal() function.

[2.0.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.4, 10.7, 12.2.

[2.0.2]

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

[2.0.1]

- Support channel mux setting in INPUTMUX_EnableSignal().

[2.0.0]

- Initial version.
-

IOCON

[2.2.0]

- Improvements
 - Removed duplicate macro definitions.
 - Renamed 'IOCON_I2C_SLEW' macro to 'IOCON_I2C_MODE' to match its companion 'IOCON_GPIO_MODE'. The original is kept as a deprecated symbol.

[2.1.2]

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

[2.1.1]

- Updated left shift format with mask value instead of a constant value to automatically adapt to all platforms.

[2.1.0]

- Added a new IOCON_PinMuxSet() function with a feature IOCON_ONE_DIMENSION for LPC845MAX board.

[2.0.0]

- Initial version.
-

IPED

[2.2.0]

- Renamed CSS to ELS.

[2.1.1]

- Fix build error due to renamed symbols.

[2.1.0]

- Add IPED_Config() (including FFR CMPA write) and IPED_Reconfig() features.

[2.0.0]

- Initial version.
-

IRTC

[2.3.3]

- Bug Fixes
 - Fix CERT INT31-C issue.

[2.3.2]

- Bug Fixes
 - Fixed API IRTC_GetDatetime read YEARMON, DAYS, HOURMIN, SECONDS registers issue.

[2.3.1]

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

[2.3.0]

- New Feature
 - Supported platforms with multiple IRTC instances.

[2.2.4]

- Bug Fixes
 - Fixed MISRA C-2012 issue 10.1, 10.3, 10.4, 10.7, 12.2.

[2.2.3]

- Bug Fixes
 - Updated undefined macro names by available ones.

[2.2.2]

- Bug Fixes
 - Fixed MISRA C-2012 issue 10.3.

[2.2.1]

- Bug Fixes
 - Fixed MISRA issues.

[2.2.0]

- New Feature
 - Add new APIs for CLK_SEL and CLKO to select RTC clock and enable/disable output to peripherals.
 - Supported platforms without tamper feature.

[2.1.3]

- Bug Fixes
 - Fixed MISRA C-2012 issue 10.1 and 10.4.

[2.1.2]

- Bug Fixes
 - Fixed kIRTC_TamperFlag flag can't be cleared issue.

[2.1.1]

- Bug Fixes
 - MISRA C-2012 issue check.
 - * Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4.

[2.1.0]

- Bug Fixes
 - Fixed incorrect leap year check in IRTC_CheckDatetimeFormat.
- New Feature
 - Added new APIs for new feature FSL_FEATURE_RTC_HAS_SUBSYSTEM.
 - Added new APIs for TAMPER, TAMPER QUEUE status get and clear.
 - Added new API to enable/disable 32 kHz RTC OSC clock during RTC register write.
 - Updated IRTC_SetTamperParams to support new feature FSL_FEATURE_RTC_HAS_FILTER23_CFG
 - Updated irtc_config_t to exclude member wakeupSelect for new feature FSL_FEATURE_RTC_HAS_NO_CTRL2_WAKEUP_MODE.

[2.0.2]

- Bug Fixes
 - MISRA C-2012 issue check.
 - * Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.6, rule-10.8, rule-11.9, rule-12.2, rule-15.5, rule-16.4, rule-17.7.

[2.0.1]

- Bug Fixes
 - Fixed the issue of hard code in IRTC_Init.

[2.0.0]

- Initial version.
-

ITRC**[2.4.0]**

- Rework the input signal definition for better flexibility

[2.3.0]

- Update names of kITRC_SwEvent1/2 to kITRC_SwEvent0/1 to align with RM

[2.2.0]

- Update driver to new version and input events

[2.1.0]

- Make SYSCON glitch platform dependent.

[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.
-

MCAN

[2.4.2]

- Bug Fixes
 - Fixed MISRA issue rule-10.3, rule-10.6, rule-10.7 and rule-15.7.

[2.4.1]

- Bug Fixes
 - Fixed incorrect fifo1 status on message lost.

[2.4.0]

- Improvements
 - Add MCAN_CalculateSpecifiedTimingValues() API to get CAN bit timing parameter with user-defined settings.
 - Add MCAN_FDCalculateSpecifiedTimingValues() API to get CANFD bit timing parameter with user-defined settings.

[2.3.2]

- Bug Fixes
 - Fix MISRA C-2012 issue 10.1 and 10.4.

[2.3.1]

- Bug Fixes
 - Fixed the issue that MCAN_TransferSendNonBlocking() API can't send remote frame.

[2.3.0]

- Improvements
 - Add MCAN_SetMessageRamConfig() API to perform global message RAM configure.
 - Add MCAN_EnterInitialMode() API.

[2.2.0]

- Improvements
 - Add MCAN_SetBaudRate/MCAN_SetBaudRateFD APIs to make users easy to set CAN baud rate.

[2.1.8]

- Bug Fixes
 - Add check FIFO status code in MCAN_ReadRxFifo() to avoid read back empty frame and wrong trigger the FIFO index increase.

[2.1.7]

- Bug Fixes
 - Fixed the clear error flags issue in MCAN_TransferHandleIRQ() API.
 - Fixed the Solve Tx interrupt issue in MCAN_TransferHandleIRQ() API which may abort the unhandled transfers.
 - Remove disable global tx interrupt from MCAN_TransferAbortSend API.

[2.1.6]

- Bug Fixes
 - Fixed the issue of writing 1 in the following functions.
 - MCAN_TransmitAddRequest
 - MCAN_TransmitCancelRequest
 - MCAN_ClearRxBufferStatusFlag

[2.1.5]

- Bug Fixes
 - Fix MISRA C-2012 issue.

[2.1.4]

- Improvements
 - Updated improve timing APIs to make it can calculate the CiA recommended timing configuration.
 - Implement Transmitter Delay Compensation feature.
 - Modify the default baudRateFD value to 2M.
- Bug Fixes
 - Fixed the code error issue in MCAN_ClearStatusFlag() to avoid clear all flags.

[2.1.3]

- Bug Fixes
 - Fixed the code error issue and simplified the algorithm in improved timing APIs.
 - * MCAN_CalculateImprovedTimingValues
 - * MCAN_FDCalculateImprovedTimingValues

[2.1.2]

- Bug Fixes
 - Fixed the non-divisible case in improved timing APIs.
 - * MCAN_CalculateImprovedTimingValues
 - * MCAN_FDCalculateImprovedTimingValues

[2.1.1]

- Bug Fixes
 - MISRA C-2012 issue check.
 - * Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.6, rule-10.7, rule-10.8, rule-11.9, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-17.7, rule-18.4, rule-2.2, rule-21.15, rule-5.8, rule-8.3.
 - * Fixed the Coverity issue of BAD_SHIFT in MCAN.
 - * Fixed the issue of Pa082 warning.
 - * Fixed the issue of dropping interrupt flags in handler function.

[2.1.0]

- Bug Fixes
 - Fixed Coverity issue FORWARD_NULL.
 - Fixed Clang issue.
 - Fixed legacy issue in the driver and changed default bus data baud rate for CANFD.
- Improvements
 - Implemented feature for improved timing configuration.

[2.0.3]

- Improvements
 - Used memset to initialize the structure before using.
 - Added function definition comment in c file.
 - Updated source file license to SPDX BSD_3.
 - Corrected capital mistake of Fifo and fifo.
 - Reset the MCAN module in LPC drivers after clock enable.

[2.0.2]

- Bug Fixes
 - Picked MISRA fixed in release 8 branch.
 - MISRA C 2012 fixed regarding FlexCAN and MCAN address update.
- Improvements
 - Implemented for delay/retry in MCAN driver.

[2.0.1]

- Improvements
 - LPC54608 chip did not support the FD feature, so added a feature macro for it.

[2.0.0]

- Initial version.
-

MRT**[2.0.5]**

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.0.4]

- Improvements
 - Don't reset MRT when there is not system level MRT reset functions.

[2.0.3]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.1 and 10.4.
 - Fixed the wrong count value assertion in MRT_StartTimer API.

[2.0.2]

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

[2.0.1]

- Added control macro to enable/disable the RESET and CLOCK code in current driver.

[2.0.0]

- Initial version.
-

OPAMP

[2.2.1]

- Improvements
 - Fixed CERT-C issues.

[2.2.0]

- Improvements
 - Added OPAMP positive port gain and negative gain configuration functions.

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- Improvements
 - Updated opamp driver code, supports devices MCXN947 and MCXN548.
 - Added OPAMP_EnableRefBuffer() API to control bias voltage input buffer.
 - Added OPAMP_EnableTriggerMode() API to control OPAMP trigger mode.

[2.0.0]

- Initial version.
-

OSTIMER

[2.2.6]

- Improvements
 - Drop the check of MATCH_WR_RDY and ostimer counter value in OSTIMER_SetMatchRawValue. In most applications, they are useless and may bring at least 7 OSTimer ticks latency, which is unacceptable when OSTimer is working under slow peripheral clock.
 - Optimize software gray code to binary conversion: replaced loop-based implementation with branchless bitwise operations.

[2.2.5]

- Improvements
 - Support binary encoded ostimer.

[2.2.4]

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.2.3]

- Improvements
 - Disable and clear pending interrupts before disabling the OSTIMER clock to avoid interrupts being executed when the clock is already disabled.

[2.2.2]

- Improvements
 - Support devices with different OSTIMER instance name.

[2.2.1]

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

[2.2.0]

- Improvements
 - Move the PMC operation out of the OSTIMER driver to board specific files.
 - Added low level APIs to control OSTIMER MATCH and interrupt.

[2.1.2]

- Bug Fixes
 - Fixed MISRA-2012 rule 10.8.

[2.1.1]

- Bug Fixes
 - removes the suffix 'n' for some register names and bit fields' names
- Improvements
 - Added HW CODE GRAY feature supported by CODE GRAY in SYSCTRL register group.

[2.1.0]

- Bug Fixes
 - Added a workaround to fix the issue that no interrupt was reported when user set smaller period.
 - Fixed violation of MISRA C-2012 rule 10.3 and 11.9.
- Improvements
 - Added return value for the two APIs to set match value.
 - * OSTIMER_SetMatchRawValue

* OSTIMER_SetMatchValue

[2.0.3]

- Bug Fixes
 - Fixed violation of MISRA C-2012 rule 10.3, 14.4, 17.7.

[2.0.2]

- Improvements
 - Added support for OSTIMER0

[2.0.1]

- Improvements
 - Removed the software reset function out of the initialization API.
 - Enabled interrupt directly instead of enabling deep sleep interrupt. Users need to enable the deep sleep interrupt in application code if needed.

[2.0.0]

- Initial version.
-

PINT

[2.2.0]

- Fixed
 - Fixed the issue that clear interrupt flag when it's not handled. This causes events to be lost.
- Changed
 - Used one callback for one PINT instance. It's unnecessary to provide different callbacks for all PINT events.

[2.1.13]

- Improvements
 - Added instance array for PINT to adapt more devices.
 - Used release reset instead of reset PINT which may clear other related registers out of PINT.

[2.1.12]

- Bug Fixes
 - Fixed coverity issue.

[2.1.11]

- Bug Fixes
 - Fixed MISRA C-2012 rule 10.7 violation.

[2.1.10]

- New Features
 - Added the driver support for MCXN10 platform with combined interrupt handler.

[2.1.9]

- Bug Fixes
 - Fixed MISRA-2012 rule 8.4.

[2.1.8]

- Bug Fixes
 - Fixed MISRA-2012 rule 10.1 rule 10.4 rule 10.8 rule 18.1 rule 20.9.

[2.1.7]

- Improvements
 - Added fully support for the SECPINT, making it can be used just like PINT.

[2.1.6]

- Bug Fixes
 - Fixed the bug of not enabling common pint clock when enabling security pint clock.

[2.1.5]

- Bug Fixes
 - Fixed issue for MISRA-2012 check.
 - * Fixed rule 10.1 rule 10.3 rule 10.4 rule 10.8 rule 14.4.
 - Changed interrupt init order to make pin interrupt configuration more reasonable.

[2.1.4]

- Improvements
 - Added feature to control distinguish PINT/SECPINT relevant interrupt/clock configurations for PINT_Init and PINT_Deinit API.
 - Swapped the order of clearing PIN interrupt status flag and clearing pending NVIC interrupt in PINT_EnableCallback and PINT_EnableCallbackByIndex function.
 - Bug Fixes
 - * Fixed build issue caused by incorrect macro definitions.

[2.1.3]

- Bug fix:
 - Updated PINT_PinInterruptClrStatus to clear PINT interrupt status when the bit is asserted and check whether was triggered by edge-sensitive mode.
 - Write 1 to IST corresponding bit will clear interrupt status only in edge-sensitive mode and will switch the active level for this pin in level-sensitive mode.
 - Fixed MISRA c-2012 rule 10.1, rule 10.6, rule 10.7.
 - Added FSL_FEATURE_SECPINT_NUMBER_OF_CONNECTED_OUTPUTS to distinguish IRQ relevant array definitions for SECPINT/PINT on lpc55s69 board.
 - Fixed PINT driver c++ build error and remove index offset operation.

[2.1.2]

- Improvement:
 - Improved way of initialization for SECPINT/PINT in PINT_Init API.

[2.1.1]

- Improvement:
 - Enabled secure pint interrupt and add secure interrupt handle.

[2.1.0]

- Added PINT_EnableCallbackByIndex/PINT_DisableCallbackByIndex APIs to enable/disable callback by index.

[2.0.2]

- Added control macro to enable/disable the RESET and CLOCK code in current driver.

[2.0.1]

- Bug fix:
 - Updated PINT driver to clear interrupt only in Edge sensitive.

[2.0.0]

- Initial version.
-

Power

[2.0.1]

- Update BoD value when switch clock frequency.

[2.0.0]

- Initial version.
-

POWERQUAD**[2.2.1]**

- Bug Fixes
 - Fixed CERT-C issues.

[2.2.0]

- New Features
 - Added new API PQ_Arctan2Fixed.

[2.1.1]

- Bug Fixes
 - Remove PQ_WaitDone from PQ_ArctanFixed and PQ_ArctanhFixed because it is unnecessary.

[2.1.0]

- Improvements
 - Fixed typo issue for biquad related function name.
 - Changed operator from “%” into “&” to reduce heavy cycle for biquad functions.

[2.0.5]

- Improvements
 - Added a note in driver for FIR that powerquad has a hardware limitation, when using it for FIR increment calculation, the address of pSrc needs to be a continuous address.

[2.0.4]

- Improvements
 - Supported the platforms which don't have PowerQuad clock and reset control.

[2.0.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.3, 10.4, 10.6, and so on.

[2.0.2]

- Bug Fixes
 - Fixed array size issue in fsl_powerquad_data.h file.
 - Fixed vector function pipeline issue.

[2.0.1]

- Bug Fixes
 - Fixed build error in C++ mode.

[2.0.0]

- Initial version.
-

PRINCE

- Version 2.6.0
- Renamed CSS to ELS.
 - Version 2.5.1
- Fix build error due to renamed symbols.
 - Version 2.3.2
- Fix documentation of enumeration.
- Extend PRINCE example.
 - Version 2.3.1
- Fix MISRA-2012 issues.
- Add support for LPC55S0x series
 - Version 2.3.0
- Add support for LPC55S1x and LPC55S2x series
 - Version 2.2.0
- Add runtime checking of the A0 and A1 rev. of LPC55Sxx serie to support both silicone revisions.
 - Version 2.1.0
- Update for the A1 rev. of LPC55Sxx serie.

[2.0.0]

- Initial version.
-

PUFv3

[2.0.4]

- Fix warning.

[2.0.3]

- Update for various PUF CTRL wrapper.

[2.0.2]

- Fix MISRA issue in driver.

[2.0.1]

- Fix PUF initialization issue and update driver to reflect SoC header changes.

[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_SetOouputToIdle 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.
-

RESET

[2.4.0]

- Improvements
 - Add RESET_ReleasePeripheralReset API.

[2.0.0]

- Initial version.
-

SCTIMER

[2.5.1]

- Bug Fixes
 - Fixed bug in SCTIMER_SetupCaptureAction: When kSCTIMER_Counter_H is selected, events 12-15 and capture registers 12-15 CAPn_H field can't be used.

[2.5.0]

- Improvements
 - Add SCTIMER_GetCaptureValue API to get capture value in capture registers.

[2.4.9]

- Improvements
 - Supported platforms which don't have system level SCTIMER reset.

[2.4.8]

- Bug Fixes
 - Fixed the issue that the SCTIMER_UpdatePwmDutycycle() can't writes MATCH_H bit and RELOADn_H.

[2.4.7]

- Bug Fixes
 - Fixed the issue that the SCTIMER_UpdatePwmDutycycle() can't configure 100% duty cycle PWM.

[2.4.6]

- Bug Fixes
 - Fixed the issue where the H register was not written as a word along with the L register.
 - Fixed the issue that the SCTIMER_SetCOUNTValue() is not configured with high 16 bits in unify mode.

[2.4.5]

- Bug Fixes
 - Fix SCT_EV_STATE_STATEMSKn macro build error.

[2.4.4]

- Bug Fixes
 - Fix MISRA C-2012 issue 10.8.

[2.4.3]

- Bug Fixes
 - Fixed the wrong way of writing CAPCTRL and REGMODE registers in SCTIMER_SetupCaptureAction.

[2.4.2]

- Bug Fixes
 - Fixed SCTIMER_SetupPwm 100% duty cycle issue.

[2.4.1]

- Bug Fixes
 - Fixed the issue that MATCHn_H bit and RELOADn_H bit could not be written.

[2.4.0]

[2.3.0]

- Bug Fixes
 - Fixed the potential overflow issue of pulseperiod variable in SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle API.
 - Fixed the issue of SCTIMER_CreateAndScheduleEvent API does not correctly work with 32 bit unified counter.
 - Fixed the issue of position of clear counter operation in SCTIMER_Init API.
- Improvements
 - Update SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle to support generate 0% and 100% PWM signal.
 - Add SCTIMER_SetupEventActiveDirection API to configure event activity direction.
 - Update SCTIMER_StartTimer/SCTIMER_StopTimer API to support start/stop low counter and high counter at the same time.
 - Add SCTIMER_SetCounterState/SCTIMER_GetCounterState API to write/read counter current state value.
 - Update APIs to make it meaningful.
 - * SCTIMER_SetEventInState
 - * SCTIMER_ClearEventInState
 - * SCTIMER_GetEventInState

[2.2.0]

- Improvements
 - Updated for 16-bit register access.

[2.1.3]

- Bug Fixes
 - Fixed the issue of uninitialized variables in SCTIMER_SetupPwm.
 - Fixed the issue that the Low 16-bit and high 16-bit work independently in SCTIMER driver.
- Improvements
 - Added an enumerable macro of unify counter for user.
 - * kSCTIMER_Counter_U
 - Created new APIs for the RTC driver.
 - * SCTIMER_SetupStateLdMethodAction
 - * SCTIMER_SetupNextStateActionwithLdMethod
 - * SCTIMER_SetCOUNTValue
 - * SCTIMER_GetCOUNTValue
 - * SCTIMER_SetEventInState
 - * SCTIMER_ClearEventInState

- * SCTIMER_GetEventInState
- Deprecated legacy APIs for the RTC driver.
- * SCTIMER_SetupNextStateAction

[2.1.2]

- Bug Fixes
 - MISRA C-2012 issue fixed: rule 10.3, 10.4, 10.6, 10.7, 11.9, 14.2 and 15.5.

[2.1.1]

- Improvements
 - Updated the register and macro names to align with the header of devices.

[2.1.0]

- Bug Fixes
 - Fixed issue where SCT application level Interrupt handler function is occupied by SCT driver.
 - Fixed issue where wrong value for INSYNC field inside SCTIMER_Init function.
 - Fixed issue to change Default value for INSYNC field inside SCTIMER_GetDefaultConfig.

[2.0.1]

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

[2.0.0]

- Initial version.
-

SPI

[2.3.2]

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

[2.3.1]

- Improvements
 - Changed SPI_DUMMYDATA to 0x00.

[2.3.0]

- Update version.

[2.2.2]

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

[2.2.1]

- Bug Fixes
 - Fixed MISRA 2012 10.4 issue.
 - Added code to clear FIFOs before transfer using DMA.

[2.2.0]

- Bug Fixes
 - Fixed bug that slave gets stuck during interrupt transfer.

[2.1.1]

- Improvements
 - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
 - Fixed MISRA 10.1, 5.7 issues.

[2.1.0]

- Bug Fixes
 - Fixed Coverity issue of incrementing null pointer in SPI_TransferHandleIRQInternal.
 - Eliminated IAR Pa082 warnings.
 - Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.
- New Features
 - Modified the definition of SPI_SSELPOL_MASK to support the socs that have only 3 SSEL pins.

[2.0.4]

- Bug Fixes
 - Fixed the bug of using read only mode in DMA transfer. In DMA transfer mode, if transfer->txData is NULL, code attempts to read data from the address of 0x0 for configuring the last frame.
 - Fixed wrong assignment of handle->state. During transfer handle->state should be kSPI_Busy rather than kStatus_SPI_Busy.
- Improvements
 - Rounded up the calculated divider value in SPI_MasterSetBaud.

[2.0.3]

- Improvements
 - Added “SPI_FIFO_DEPTH(base)” with more definition.

[2.0.2]

- Improvements
 - Unified the component full name to FLEXCOMM SPI(DMA/FREERTOS) driver.

[2.0.1]

- Changed the data buffer from uint32_t to uint8_t which matches the real applications for SPI DMA driver.
- Added dummy data setup API to allow users to configure the dummy data to be transferred.
- Added new APIs for half-duplex transfer function. Users can not only send and receive data by one API in polling/interrupt/DMA way, but choose either to transmit first or to receive first. Besides, the PCS pin can be configured as assert status in transmission (between transmit and receive) by setting the isPcsAssertInTransfer to true.

[2.0.0]

- Initial version.
-

SPI_DMA**[2.2.2]**

- Bug Fixes
 - Fixed the bug half duplex mode can't be used if data size is larger than 1024 bytes.

[2.2.1]

- Bug Fixes
 - Fixed MISRA 2012 11.6 issue..

[2.2.0]

- Improvements
 - Supported dataSize larger than 1024 data transmit.
-

SYSCTL**[2.0.5]**

- Bug Fixes:
 - Fixed violations of MISRA C-2012 rule 8.3, 10.1, 10.4, 10.7.

[2.0.4]

- Improvements:
 - Update macro name to align with the header of devices.

[2.0.3]

- Improvements:
 - Update the register and macro name to align with the header of devices.

[2.0.2]

- Removed kSYSCTL_Flexcomm3DataOut enumeration definition.

[2.0.1]

- Fixed some typo error comments and improved driver integral ability.

[2.0.0]

- Initial version.
-

USART

[2.8.5]

- Bug Fixes
 - Fixed race condition during call of USART_EnableTxDMA and USART_EnableRxDMA.

[2.8.4]

- Bug Fixes
 - Fixed exclusive access in USART_TransferReceiveNonBlocking and USART_TransferSendNonBlocking.

[2.8.3]

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

[2.8.2]

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

[2.8.1]

- Bug Fixes
 - Fixed the Baud Rate Generator(BRG) configuration in 32kHz mode.

[2.8.0]

- New Features
 - Added the rx timeout interrupts and status flags of bus status.
 - Added new rx timeout configuration item in `usart_config_t`.
 - Added API `USART_SetRxTimeoutConfig` for rx timeout configuration.
- Improvements
 - When the calculated baudrate cannot meet user's configuration, lower OSR value is allowed to use.

[2.7.0]

- New Features
 - Added the missing interrupts and status flags of bus status.
 - Added the check of tx error, noise error framing error and parity error in interrupt handler.

[2.6.0]

- Improvements
 - Used separate data for TX and RX in `usart_transfer_t`.
- Bug Fixes
 - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling `USART_TransferReceiveNonBlocking`, the received data count returned by `USART_TransferGetReceiveCount` is wrong.
- New Features
 - Added missing API `USART_TransferGetSendCountDMA` get send count using DMA.

[2.5.0]

- New Features
 - Added APIs `USART_GetRxFifoCount`/`USART_GetTxFifoCount` to get rx/tx FIFO data count.
 - Added APIs `USART_SetRxFifoWatermark`/`USART_SetTxFifoWatermark` to set rx/tx FIFO water mark.
- Bug Fixes
 - Fixed DMA transfer blocking issue by enabling tx idle interrupt after DMA transmission finishes.

[2.4.0]

- New Features
 - Modified `usart_config_t`, `USART_Init` and `USART_GetDefaultConfig` APIs so that the hardware flow control can be enabled during module initialization.
- Bug Fixes
 - Fixed MISRA 10.4 violation.

[2.3.1]

- Bug Fixes
 - Fixed bug that operation on INTENSET, INTENCLR, FIFOINTENSET and FIFOINTENCLR should use bitwise operation not ‘or’ operation.
 - Fixed bug that if rx interrupt occurs before TX interrupt is enabled and after txDataSize is configured, the data will be sent early by mistake, thus TX interrupt will be enabled after data is sent out.
- Improvements
 - Added check for baud rate’s accuracy that returns kStatus_USART_BaudrateNotSupport when the best achieved baud rate is not within 3% error of configured baud rate.

[2.3.0]

- New Features
 - Added APIs to configure 9-bit data mode, set slave address and send address.
 - Modified USART_TransferReceiveNonBlocking and USART_TransferHandleIRQ to use 9-bit mode in multi-slave system.

[2.2.0]

- New Features
 - Added the feature of supporting USART working at 32 kHz clocking mode.
- Improvements
 - Modified USART_TransferHandleIRQ so that txState will be set to idle only when all data has been sent out to bus.
 - Modified USART_TransferGetSendCount so that this API returns the real byte count that USART has sent out rather than the software buffer status.
 - Added timeout mechanism when waiting for certain states in transfer driver.
- Bug Fixes
 - Fixed MISRA 10.1 issues.
 - Fixed bug that operation on INTENSET, INTENCLR, FIFOINTENSET and FIFOINTENCLR should use bitwise operation not ‘or’ operation.
 - Fixed bug that if rx interrupt occurs before TX interrupt is enabled and after txDataSize is configured, the data will be sent early by mistake, thus TX interrupt will be enabled after data is sent out.

[2.1.1]

- Improvements
 - Added check for transmitter idle in USART_TransferHandleIRQ and USART_TransferSendDMACallback to ensure all the data would be sent out to bus.
 - Modified USART_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.
- Bug Fixes
 - Eliminated IAR Pa082 warnings.

- Fixed MISRA issues.
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.

[2.1.0]

- New Features
 - Added features to allow users to configure the USART to synchronous transfer(master and slave) mode.
- Bug Fixes
 - Modified USART_SetBaudRate to get more accurate configuration.

[2.0.3]

- New Features
 - Added new APIs to allow users to enable the CTS which determines whether CTS is used for flow control.

[2.0.2]

- Bug Fixes
 - Fixed the bug where transfer abort APIs could not disable the interrupts. The FIFOINTENSET register should not be used to disable the interrupts, so use the FIFOINTENCLR register instead.

[2.0.1]

- Improvements
 - Unified the component full name to FLEXCOMM USART (DMA/FREERTOS) driver.

[2.0.0]

- Initial version.
-

USART_DMA

[2.6.0]

- Refer USART driver change log 2.0.1 to 2.6.0
-

UTICK

[2.0.5]

- Improvements
 - Improved for SOC RW610.

[2.0.4]

- Bug Fixes
 - Fixed compile fail issue of no-supporting PD configuration in utick driver.

[2.0.3]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rules: 8.4, 14.4, 17.7

[2.0.2]

- Added new feature definition macro to enable/disable power control in drivers for some devices have no power control function.

[2.0.1]

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

[2.0.0]

- Initial version.
-

VREF

[2.4.0]

- Improvements
 - Support TEST_UNLOCK and TRIM0 registers read and write.

[2.3.0]

- Improvements
 - Add feature macros for compatibility with some platforms that don't have CSR[LPBG_BUF_EN] and UTRIM[TRIM2V1].

[2.2.2]

- Bug Fixes
 - Fixed typo in vref_buffer_mode_t enumerator.

[2.2.1]

- Improvements
 - Updated VREF driver code to reduce code redundancy.

[2.2.0]

- Improvements
 - Remove API VREF_SetTrimVal and VREF_GetTrimVal
 - Add new API VREF_SetVrefTrimVal,VREF_SetTrim21Val,VREF_GetVrefTrimVal and VREF_GetTrim21Val
 - Updated VREF driver code to conform to VREF IP architecture.

[2.1.0]

- Improvements
 - Remove API VREF_SetTrimVal and VREF_GetTrimVal
 - Add new API VREF_SetVrefTrimVal,VREF_SetTrim21Val,VREF_GetVrefTrimVal and VREF_GetTrim21Val
 - Updated VREF driver code to conform to VREF IP architecture.
- Bug Fixes
 - Fix issue which progeam stuck in VREF_Init

[2.0.0]

- Initial version.
-

WWDT**[2.1.10]**

- Bug Fixes
 - Chek WWDT_RSTS instead of FSL_FEATURE_WWDT_HAS_NO_RESET to determine whether the peripheral can be reset.

[2.1.9]

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

[2.1.8]

- Improvements
 - Updated the “WWDT_Init” API to add wait operation. Which can avoid the TV value read by CPU still be 0xFF (reset value) after WWDT_Init function returns.

[2.1.7]

- Bug Fixes
 - Fixed the issue that the watchdog reset event affected the system from PMC.
 - Fixed the issue of setting watchdog WDPROTECT field without considering the backwards compatibility.

- Fixed the issue of clearing bit fields by mistake in the function of WWDT_ClearStatusFlags.

[2.1.5]

- Bug Fixes
 - deprecated a unusable API in WWDT driver.
 - * WWDT_Disable

[2.1.4]

- Bug Fixes
 - Fixed violation of the MISRA C-2012 rules Rule 10.1, 10.3, 10.4 and 11.9.
 - Fixed the issue of the inseparable process interrupted by other interrupt source.
 - * WWDT_Init

[2.1.3]

- Bug Fixes
 - Fixed legacy issue when initializing the MOD register.

[2.1.2]

- Improvements
 - Updated the “WWDT_ClearStatusFlags” API and “WWDT_GetStatusFlags” API to match QN9090. WDTOF is not set in case of WD reset. Get info from PMC instead.

[2.1.1]

- New Features
 - Added new feature definition macro for devices which have no LCOK control bit in MOD register.
 - Implemented delay/retry in WWDT driver.

[2.1.0]

- Improvements
 - Added new parameter in configuration when initializing WWDT module. This parameter, which must be set, allows the user to deliver the WWDT clock frequency.

[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.

[*LPC55S36*](#)

1.7 Middleware Documentation

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

1.7.1 Multicore

[*multicore*](#)

1.7.2 Audio Voice components

[*components*](#)

1.7.3 Maestro Audio Framework for MCU

[*Maestro Audio Framework*](#)

1.7.4 FreeMASTER

[*freemaster*](#)

1.7.5 FreeRTOS

[*FreeRTOS*](#)

1.7.6 File systemFatfs

[*fatfs*](#)

Chapter 2

LPC55S36

2.1 ANACTRL: Analog Control Driver

`void ANACTRL_Init(ANACTRL_Type *base)`

Initializes the ANACTRL mode, the module's clock will be enabled by invoking this function.

Parameters

- `base` – ANACTRL peripheral base address.

`void ANACTRL_Deinit(ANACTRL_Type *base)`

De-initializes ANACTRL module, the module's clock will be disabled by invoking this function.

Parameters

- `base` – ANACTRL peripheral base address.

`void ANACTRL_SetFro192M(ANACTRL_Type *base, const anactrl_fro192M_config_t *config)`

Configures the on-chip high-speed Free Running Oscillator(FRO192M), such as enabling/disabling 12 MHz clock output and enable/disable 96MHz clock output.

Parameters

- `base` – ANACTRL peripheral base address.
- `config` – Pointer to FRO192M configuration structure. Refer to `anactrl_fro192M_config_t` structure.

`void ANACTRL_GetDefaultFro192MConfig(anactrl_fro192M_config_t *config)`

Gets the default configuration of FRO192M. The default values are:

```
config->enable12MHzClk = true;  
config->enable96MHzClk = false;
```

Parameters

- `config` – Pointer to FRO192M configuration structure. Refer to `anactrl_fro192M_config_t` structure.

`void ANACTRL_SetXo32M(ANACTRL_Type *base, const anactrl_xo32M_config_t *config)`

Configures the 32 MHz Crystal oscillator(High-speed crystal oscillator), such as enable/disable output to CPU system, and so on.

Parameters

- `base` – ANACTRL peripheral base address.
- `config` – Pointer to XO32M configuration structure. Refer to `anactrl_xo32M_config_t` structure.

`void ANACTRL_GetDefaultXo32MConfig(anactrl_xo32M_config_t *config)`

Gets the default configuration of XO32M. The default values are:

```
config->enableSysClkOutput = false;  
config->enableACBufferBypass = false;
```

Parameters

- `config` – Pointer to XO32M configuration structure. Refer to `anactrl_xo32M_config_t` structure.

`static inline void ANACTRL_EnableInterrupts(ANACTRL_Type *base, uint32_t mask)`

Enables the ANACTRL interrupts.

Parameters

- `base` – ANACTRL peripheral base address.
- `mask` – The interrupt mask. Refer to “`_anactrl_interrupt`” enumeration.

`static inline void ANACTRL_DisableInterrupts(ANACTRL_Type *base, uint32_t mask)`

Disables the ANACTRL interrupts.

Parameters

- `base` – ANACTRL peripheral base address.
- `mask` – The interrupt mask. Refer to “`_anactrl_interrupt`” enumeration.

`static inline void ANACTRL_ClearInterrupts(ANACTRL_Type *base, uint32_t mask)`

Clears the ANACTRL interrupts.

Parameters

- `base` – ANACTRL peripheral base address.
- `mask` – The interrupt mask. Refer to “`_anactrl_interrupt`” enumeration.

`static inline uint32_t ANACTRL_GetStatusFlags(ANACTRL_Type *base)`

Gets ANACTRL status flags.

This function gets Analog control status flags. The flags are returned as the logical OR value of the enumerators `_anactrl_flags`. To check for a specific status, compare the return value with enumerators in the `_anactrl_flags`. For example, to check whether the flash is in power down mode:

```
if (kANACTRL_FlashPowerDownFlag & ANACTRL_ANACTRL_GetStatusFlags(ANACTRL))  
{  
    ...  
}
```

Parameters

- `base` – ANACTRL peripheral base address.

Returns

ANACTRL status flags which are given in the enumerators in the `_anactrl_flags`.

static inline uint32_t ANACTRL_GetOscStatusFlags(ANACTRL_Type *base)

Gets ANACTRL oscillators status flags.

This function gets Anactrl oscillators status flags. The flags are returned as the logical OR value of the enumerators `_anactrl_osc_flags`. To check for a specific status, compare the return value with enumerators in the `_anactrl_osc_flags`. For example, to check whether the FRO192M clock output is valid:

```
if (kANACTRL_OutputClkValidFlag & ANACTRL_ANACTRL_GetOscStatusFlags(ANACTRL))
{
    ...
}
```

Parameters

- `base` – ANACTRL peripheral base address.

Returns

ANACTRL oscillators status flags which are given in the enumerators in the `_anactrl_osc_flags`.

static inline uint32_t ANACTRL_GetInterruptStatusFlags(ANACTRL_Type *base)

Gets ANACTRL interrupt status flags.

This function gets Anactrl interrupt status flags. The flags are returned as the logical OR value of the enumerators `_anactrl_interrupt_flags`. To check for a specific status, compare the return value with enumerators in the `_anactrl_interrupt_flags`. For example, to check whether the VBAT voltage level is above the threshold:

```
if (kANACTRL_BodVbatPowerFlag & ANACTRL_ANACTRL_GetInterruptStatusFlags(ANACTRL))
{
    ...
}
```

Parameters

- `base` – ANACTRL peripheral base address.

Returns

ANACTRL oscillators status flags which are given in the enumerators in the `_anactrl_osc_flags`.

static inline void ANACTRL_EnableVref1V(ANACTRL_Type *base, bool enable)

Aux_Bias Control Interfaces.

Enables/disables 1V reference voltage buffer.

Parameters

- `base` – ANACTRL peripheral base address.
- `enable` – Used to enable or disable 1V reference voltage buffer.

enum `_anactrl_interrupt_flags`

ANACTRL interrupt flags.

Values:

enumerator `kANACTRL_BodVbatFlag`

BOD VBAT Interrupt status before Interrupt Enable.

enumerator `kANACTRL_BodVbatInterruptFlag`

BOD VBAT Interrupt status after Interrupt Enable.

enumerator kANACTRL__BodVbatPowerFlag

Current value of BOD VBAT power status output.

enumerator kANACTRL__BodCoreFlag

BOD CORE Interrupt status before Interrupt Enable.

enumerator kANACTRL__BodCoreInterruptFlag

BOD CORE Interrupt status after Interrupt Enable.

enumerator kANACTRL__BodCorePowerFlag

Current value of BOD CORE power status output.

enumerator kANACTRL__DcdcFlag

DCDC Interrupt status before Interrupt Enable.

enumerator kANACTRL__DcdcInterruptFlag

DCDC Interrupt status after Interrupt Enable.

enumerator kANACTRL__DcdcPowerFlag

Current value of DCDC power status output.

enum __anactrl__interrupt

ANACTRL interrupt control.

Values:

enumerator kANACTRL__BodVbatInterruptEnable

BOD VBAT interrupt control.

enumerator kANACTRL__BodCoreInterruptEnable

BOD CORE interrupt control.

enumerator kANACTRL__DcdcInterruptEnable

DCDC interrupt control.

enum __anactrl__flags

ANACTRL status flags.

Values:

enumerator kANACTRL__FlashPowerDownFlag

Flash power-down status.

enumerator kANACTRL__FlashInitErrorFlag

Flash initialization error status.

enum __anactrl__osc_flags

ANACTRL FRO192M and XO32M status flags.

Values:

enumerator kANACTRL__OutputClkValidFlag

Output clock valid signal.

enumerator kANACTRL__CCOThresholdVoltageFlag

CCO threshold voltage detector output (signal vcco_ok).

enumerator kANACTRL__XO32MOutputReadyFlag

Indicates XO out frequency stability.

```
typedef struct _anactrl_fro192M_config anactrl_fro192M_config_t
```

Configuration for FRO192M.

This structure holds the configuration settings for the on-chip high-speed Free Running Oscillator. To initialize this structure to reasonable defaults, call the ANACTRL_GetDefaultFro192MConfig() function and pass a pointer to your config structure instance.

```
typedef struct _anactrl_xo32M_config anactrl_xo32M_config_t
```

Configuration for XO32M.

This structure holds the configuration settings for the 32 MHz crystal oscillator. To initialize this structure to reasonable defaults, call the ANACTRL_GetDefaultXo32MConfig() function and pass a pointer to your config structure instance.

```
FSL_ANACTRL_DRIVER_VERSION
```

ANACTRL driver version.

```
struct _anactrl_fro192M_config
```

#include <fsl_anactrl.h> Configuration for FRO192M.

This structure holds the configuration settings for the on-chip high-speed Free Running Oscillator. To initialize this structure to reasonable defaults, call the ANACTRL_GetDefaultFro192MConfig() function and pass a pointer to your config structure instance.

Public Members

```
bool enable12MHzClk
```

Enable 12MHz clock.

```
bool enable96MHzClk
```

Enable 96MHz clock.

```
struct _anactrl_xo32M_config
```

#include <fsl_anactrl.h> Configuration for XO32M.

This structure holds the configuration settings for the 32 MHz crystal oscillator. To initialize this structure to reasonable defaults, call the ANACTRL_GetDefaultXo32MConfig() function and pass a pointer to your config structure instance.

Public Members

```
bool enableACBufferBypass
```

Enable XO AC buffer bypass in pll and top level.

```
bool enableSysCLKOutput
```

Enable XO 32 MHz output to CPU system, SCT, and CLKOUT

```
bool enableADCOutput
```

Enable High speed crystal oscillator output to ADC.

2.2 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

2.3 CACHE: CACHE Memory Controller

`uint32_t CACHE64_GetInstance(CACHE64_POLSEL_Type *base)`

Returns an instance number given peripheral base address.

Parameters

- `base` – The peripheral base address.

Returns

CACHE64_POLSEL instance number starting from 0.

`uint32_t CACHE64_GetInstanceByAddr(uint32_t address)`

brief Returns an instance number given physical memory address.

param address The physical memory address.

Returns

CACHE64_CTRL instance number starting from 0.

`status_t CACHE64_Init(CACHE64_POLSEL_Type *base, const cache64_config_t *config)`

Initializes an CACHE64 instance with the user configuration structure.

This function configures the CACHE64 module with user-defined settings. Call the `CACHE64_GetDefaultConfig()` function to configure the configuration structure and get the default configuration.

Parameters

- `base` – CACHE64_POLSEL peripheral base address.
- `config` – Pointer to a user-defined configuration structure.

Return values

`kStatus_Success` – CACHE64 initialize succeed

`void CACHE64_GetDefaultConfig(cache64_config_t *config)`

Gets the default configuration structure.

This function initializes the CACHE64 configuration structure to a default value. The default values are first region covers whole cacheable area, and policy set to write back.

Parameters

- `config` – Pointer to a configuration structure.

`void CACHE64_EnableCache(CACHE64_CTRL_Type *base)`

Enables the cache.

Parameters

- `base` – CACHE64_CTRL peripheral base address.

`void CACHE64_DisableCache(CACHE64_CTRL_Type *base)`

Disables the cache.

Parameters

- `base` – CACHE64_CTRL peripheral base address.

`void CACHE64_InvalidateCache(CACHE64_CTRL_Type *base)`

Invalidates the cache.

Parameters

- `base` – CACHE64_CTRL peripheral base address.

void CACHE64_InvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cache by range.

Note: Address and size should be aligned to “CACHE64_LINESIZE_BYTE”. The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be invalidated, should be larger than 0.

void CACHE64_CleanCache(CACHE64_CTRL_Type *base)

Cleans the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cache by range.

Note: Address and size should be aligned to “CACHE64_LINESIZE_BYTE”. The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be cleaned, should be larger than 0.

void CACHE64_CleanInvalidateCache(CACHE64_CTRL_Type *base)

Cleans and invalidates the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate cache by range.

Note: Address and size should be aligned to “CACHE64_LINESIZE_BYTE”. The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

```
void CACHE64_EnableWriteBuffer(CACHE64_CTRL_Type *base, bool enable)
```

Enables/disables the write buffer.

Parameters

- base – CACHE64_CTRL peripheral base address.
- enable – The enable or disable flag. true - enable the write buffer. false - disable the write buffer.

```
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 CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0.

```
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 CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0.

```
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 CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned, should be larger than 0.

```
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 CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

FSL_CACHE_DRIVER_VERSION

cache driver version.

enum _cache64_policy

Level 2 cache controller way size.

Values:

enumerator kCACHE64_PolicyNonCacheable

Non-cacheable

enumerator kCACHE64_PolicyWriteThrough

Write through

enumerator kCACHE64_PolicyWriteBack

Write back

typedef enum _cache64_policy cache64_policy_t

Level 2 cache controller way size.

typedef struct _cache64_config cache64_config_t

CACHE64 configuration structure.

CACHE64_LINESIZE_BYTE

cache line size.

CACHE64_REGION_NUM

cache region number.

CACHE64_REGION_ALIGNMENT

cache region alignment.

struct _cache64_config

#include <fsl_cache.h> CACHE64 configuration structure.

Public Members

uint32_t boundaryAddr[(3U) - 1]

< The cache controller can divide whole memory into 3 regions. Boundary address is the FlexSPI internal address (start from 0) instead of system address (start from FlexSPI AMBA base) to split adjacent regions and must be 1KB aligned. The boundary address itself locates in upper region. Cacheable policy for each region.

2.4 CACHE: LPCAC CACHE Memory Controller

`static inline void L1CACHE_EnableCodeCache(void)`
Enables the processor code bus cache.

`static inline void L1CACHE_DisableCodeCache(void)`
Disables the processor code bus cache.

`static inline void L1CACHE_InvalidateCodeCache(void)`
Clears cache.

`static inline void L1CACHE_EnableAllocation(void)`
Enables allocation.

`static inline void L1CACHE_DisableAllocation(void)`
Disables allocation.

`static inline void L1CACHE_EnableParity(void)`
Enables parity.

`static inline void L1CACHE_DisableParity(void)`
Disable parity.

`FSL_CACHE_LPCAC_DRIVER_VERSION`
cache driver version

2.5 CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing

2.6 casper_driver

`FSL_CASPER_DRIVER_VERSION`
CASPER driver version. Version 2.2.4.
Current version: 2.2.4
Change log:

- Version 2.0.0
 - Initial version
- Version 2.0.1
 - Bug fix KPSDK-24531 `double_scalar_multiplication()` result may be all zeroes for some specific input
- Version 2.0.2
 - Bug fix KPSDK-25015 `CASPER_MEMCPY` hard-fault on LPC55xx when both source and destination buffers are outside of `CASPER_RAM`
- Version 2.0.3
 - Bug fix KPSDK-28107 `RSUB`, `FILL` and `ZERO` operations not implemented in `enum_casper_operation`.
- Version 2.0.4
 - For GCC compiler, enforce `O1` optimize level, specifically to remove strict-aliasing option. This driver is very specific and requires `-fno-strict-aliasing`.

- Version 2.0.5
 - Fix sign-compare warning.
- Version 2.0.6
 - Fix IAR Pa082 warning.
- Version 2.0.7
 - Fix MISRA-C 2012 issue.
- Version 2.0.8
 - Add feature macro for CASPER_RAM_OFFSET.
- Version 2.0.9
 - Remove unused function Jac_oncurve().
 - Fix ECC384 build.
- Version 2.0.10
 - Fix MISRA-C 2012 issue.
- Version 2.1.0
 - Add ECC NIST P-521 elliptic curve.
- Version 2.2.0
 - Rework driver to support multiple curves at once.
- Version 2.2.1
 - Fix MISRA-C 2012 issue.
- Version 2.2.2
 - Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL_FEATURE_CASPER_RAM_HW_INTERLEAVE
- Version 2.2.3
 - Added macro into CASPER_Init and CASPER_Deinit to support devices without clock and reset control.
- Version 2.2.4
 - Fix MISRA-C 2012 issue.

enum __casper_operation
CASPER operation.

Values:

enumerator kCASPER_OpMul6464NoSum

enumerator kCASPER_OpMul6464Sum

Walking 1 or more of J loop, doing $r=a*b$ using $64x64=128$

enumerator kCASPER_OpMul6464FullSum

Walking 1 or more of J loop, doing $c,r=r+a*b$ using $64x64=128$, but assume inner j loop

enumerator kCASPER_OpMul6464Reduce

Walking 1 or more of J loop, doing $c,r=r+a*b$ using $64x64=128$, but sum all of w.

enumerator kCASPER_OpAdd64

Walking 1 or more of J loop, doing $c,r[-1]=r+a*b$ using $64x64=128$, but skip 1st write

enumerator kCASPER_OpSub64

Walking add with off_AB, and in/out off_RES doing $c, r = r + a + c$ using $64 + 64 = 65$

enumerator kCASPER_OpDouble64

Walking subtract with off_AB, and in/out off_RES doing $r = r - a$ using $64 - 64 = 64$, with last borrow implicit if any

enumerator kCASPER_OpXor64

Walking add to self with off_RES doing $c, r = r + r + c$ using $64 + 64 = 65$

enumerator kCASPER_OpRSub64

Walking XOR with off_AB, and in/out off_RES doing $r = r \wedge a$ using $64 \wedge 64 = 64$

enumerator kCASPER_OpShiftLeft32

Walking subtract with off_AB, and in/out off_RES using $r = a - r$

enumerator kCASPER_OpShiftRight32

Walking shift left doing $r1, r = (b * D) \mid r1$, where D is 2^{amt} and is loaded by app (off_CD not used)

enumerator kCASPER_OpCopy

Walking shift right doing $r, r1 = (b * D) \mid r1$, where D is $2^{(32 - \text{amt})}$ and is loaded by app (off_CD not used) and off_RES starts at MSW

enumerator kCASPER_OpRemask

Copy from ABoff to resoff, 64b at a time

enumerator kCASPER_OpFill

Copy and mask from ABoff to resoff, 64b at a time

enumerator kCASPER_OpZero

Fill RESOFF using 64 bits at a time with value in A and B

enumerator kCASPER_OpCompare

Fill RESOFF using 64 bits at a time of 0s

enumerator kCASPER_OpCompareFast

Compare two arrays, running all the way to the end

enum _casper_algo_t

Algorithm used for CASPER operation.

Values:

enumerator kCASPER_ECC_P256

ECC_P256

enumerator kCASPER_ECC_P384

ECC_P384

enumerator kCASPER_ECC_P521

ECC_P521

Values:

enumerator kCASPER_RamOffset_Result

enumerator kCASPER_RamOffset_Base

enumerator kCASPER_RamOffset_TempBase

enumerator kCASPER_RamOffset_Modulus

enumerator kCASPER_RamOffset_M64

typedef enum *_casper_operation* casper_operation_t
CASPER operation.

typedef enum *_casper_algo_t* casper_algo_t
Algorithm used for CASPER operation.

void CASPER_Init(CASPER_Type *base)
Enables clock and disables reset for CASPER peripheral.
Enable clock and disable reset for CASPER.

Parameters

- base – CASPER base address

void CASPER_Deinit(CASPER_Type *base)
Disables clock for CASPER peripheral.
Disable clock and enable reset.

Parameters

- base – CASPER base address

CASPER_CP

CASPER_CP_CTRL0

CASPER_CP_CTRL1

CASPER_CP_LOADER

CASPER_CP_STATUS

CASPER_CP_INTENSET

CASPER_CP_INTENCLR

CASPER_CP_INTSTAT

CASPER_CP_AREG

CASPER_CP_BREG

CASPER_CP_CREG

CASPER_CP_DREG

CASPER_CP_RES0

CASPER_CP_RES1

CASPER_CP_RES2

CASPER_CP_RES3

CASPER_CP_MASK

CASPER_CP_REMASK

CASPER_CP_LOCK

CASPER_CP_ID

CASPER_Wr32b(value, off)

CASPER_Wr64b(value, off)

CASPER_Rd32b(off)

N_wordlen_max

2.7 casper_driver_pkha

void CASPER_ModExp(CASPER_Type *base, const uint8_t *signature, const uint8_t *pubN, size_t wordLen, uint32_t pubE, uint8_t *plaintext)

Performs modular exponentiation - $(A^E) \bmod N$.

This function performs modular exponentiation.

Parameters

- base – CASPER base address
- signature – first addend (in little endian format)
- pubN – modulus (in little endian format)
- wordLen – Size of pubN in bytes
- pubE – exponent
- plaintext – **[out]** Output array to store result of operation (in little endian format)

void CASPER_ecc_init(*casper_algo_t* curve)

Initialize prime modulus mod in Casper memory .

Set the prime modulus mod in Casper memory and set N_wordlen according to selected algorithm.

Parameters

- curve – elliptic curve algorithm

void CASPER_ECC_SECP256R1_Mul(CASPER_Type *base, uint32_t resX[8], uint32_t resY[8], uint32_t X[8], uint32_t Y[8], uint32_t scalar[8])

Performs ECC secp256r1 point single scalar multiplication.

This function performs ECC secp256r1 point single scalar multiplication $[resX; resY] = scalar * [X; Y]$ Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

void CASPER_ECC_SECP256R1_MulAdd(CASPER_Type *base, uint32_t resX[8], uint32_t resY[8], uint32_t X1[8], uint32_t Y1[8], uint32_t scalar1[8], uint32_t X2[8], uint32_t Y2[8], uint32_t scalar2[8])

Performs ECC secp256r1 point double scalar multiplication.

This function performs ECC secp256r1 point double scalar multiplication $[resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2]$ Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

```
void CASPER_ECC_SECP384R1_Mul(CASPER_Type *base, uint32_t resX[12], uint32_t resY[12],  
                               uint32_t X[12], uint32_t Y[12], uint32_t scalar[12])
```

Performs ECC secp384r1 point single scalar multiplication.

This function performs ECC secp384r1 point single scalar multiplication $[resX; resY] = scalar * [X; Y]$ Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

```
void CASPER_ECC_SECP384R1_MulAdd(CASPER_Type *base, uint32_t resX[12], uint32_t  
                                resY[12], uint32_t X1[12], uint32_t Y1[12], uint32_t  
                                scalar1[12], uint32_t X2[12], uint32_t Y2[12], uint32_t  
                                scalar2[12])
```

Performs ECC secp384r1 point double scalar multiplication.

This function performs ECC secp384r1 point double scalar multiplication $[resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2]$ Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.

- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

```
void CASPER_ECC_SECP521R1_Mul(CASPER_Type *base, uint32_t resX[18], uint32_t resY[18],  
                               uint32_t X[18], uint32_t Y[18], uint32_t scalar[18])
```

Performs ECC secp521r1 point single scalar multiplication.

This function performs ECC secp521r1 point single scalar multiplication $[resX; resY] = scalar * [X; Y]$ Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

```
void CASPER_ECC_SECP521R1_MulAdd(CASPER_Type *base, uint32_t resX[18], uint32_t  
                                resY[18], uint32_t X1[18], uint32_t Y1[18], uint32_t  
                                scalar1[18], uint32_t X2[18], uint32_t Y2[18], uint32_t  
                                scalar2[18])
```

Performs ECC secp521r1 point double scalar multiplication.

This function performs ECC secp521r1 point double scalar multiplication $[resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2]$ Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

```
void CASPER_ECC_equal(int *res, uint32_t *op1, uint32_t *op2)
```

```
void CASPER_ECC_equal_to_zero(int *res, uint32_t *op1)
```

2.8 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>

2.9 Clock Driver

enum _clock_ip_name

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

enumerator kCLOCK_IpInvalid

Invalid IP name.

enumerator kCLOCK_Rom
Clock gate name: Rom.

enumerator kCLOCK_Sram1
Clock gate name: Sram1.

enumerator kCLOCK_Sram2
Clock gate name: Sram2.

enumerator kCLOCK_Sram3
Clock gate name: Sram3.

enumerator kCLOCK_Sram4
Clock gate name: Sram4.

enumerator kCLOCK_Flash
Clock gate name: Flash.

enumerator kCLOCK_Fmc
Clock gate name: Fmc.

enumerator kCLOCK_Flexspi
Clock gate name: Flexspi.

enumerator kCLOCK_InputMux
Clock gate name: InputMux.

enumerator kCLOCK_Iocon
Clock gate name: Iocon.

enumerator kCLOCK_Gpio0
Clock gate name: Gpio0.

enumerator kCLOCK_Gpio1
Clock gate name: Gpio1.

enumerator kCLOCK_Gpio2
Clock gate name: Gpio2.

enumerator kCLOCK_Gpio3
Clock gate name: Gpio3.

enumerator kCLOCK_Pint
Clock gate name: Pint.

enumerator kCLOCK_Gint
Clock gate name: Gint.

enumerator kCLOCK_Dma0
Clock gate name: Dma0.

enumerator kCLOCK_Crc0
Clock gate name: Crc.

enumerator kCLOCK_Wwdt
Clock gate name: Wwdt.

enumerator kCLOCK_Rtc0
Clock gate name: Rtc0.

enumerator kCLOCK_Mailbox
Clock gate name: Mailbox.

enumerator kCLOCK_Adc0
Clock gate name: Adc0.

enumerator kCLOCK_Adc1
Clock gate name: Adc1.

enumerator kCLOCK_Dac0
Clock gate name: Dac0.

enumerator kCLOCK_Mrt
Clock gate name: Mrt.

enumerator kCLOCK_Ostimer
Clock gate name: Ostimer.

enumerator kCLOCK_Sct
Clock gate name: Sct.

enumerator kCLOCK_Mcan
Clock gate name: Mcan.

enumerator kCLOCK_Utick
Clock gate name: Utick.

enumerator kCLOCK_FlexComm0
Clock gate name: FlexComm0.

enumerator kCLOCK_FlexComm1
Clock gate name: FlexComm1.

enumerator kCLOCK_FlexComm2
Clock gate name: FlexComm2.

enumerator kCLOCK_FlexComm3
Clock gate name: FlexComm3.

enumerator kCLOCK_FlexComm4
Clock gate name: FlexComm4.

enumerator kCLOCK_FlexComm5
Clock gate name: FlexComm5.

enumerator kCLOCK_FlexComm6
Clock gate name: FlexComm6.

enumerator kCLOCK_FlexComm7
Clock gate name: FlexComm7.

enumerator kCLOCK_MinUart0
Clock gate name: MinUart0.

enumerator kCLOCK_MinUart1
Clock gate name: MinUart1.

enumerator kCLOCK_MinUart2
Clock gate name: MinUart2.

enumerator kCLOCK_MinUart3
Clock gate name: MinUart3.

enumerator kCLOCK_MinUart4
Clock gate name: MinUart4.

enumerator kCLOCK_MinUart5
Clock gate name: MinUart5.

enumerator kCLOCK_MinUart6
Clock gate name: MinUart6.

enumerator kCLOCK_MinUart7
Clock gate name: MinUart7.

enumerator kCLOCK_LSpi0
Clock gate name: LSpi0.

enumerator kCLOCK_LSpi1
Clock gate name: LSpi1.

enumerator kCLOCK_LSpi2
Clock gate name: LSpi2.

enumerator kCLOCK_LSpi3
Clock gate name: LSpi3.

enumerator kCLOCK_LSpi4
Clock gate name: LSpi4.

enumerator kCLOCK_LSpi5
Clock gate name: LSpi5.

enumerator kCLOCK_LSpi6
Clock gate name: LSpi6.

enumerator kCLOCK_LSpi7
Clock gate name: LSpi7.

enumerator kCLOCK_BI2c0
Clock gate name: BI2c0.

enumerator kCLOCK_BI2c1
Clock gate name: BI2c1.

enumerator kCLOCK_BI2c2
Clock gate name: BI2c2.

enumerator kCLOCK_BI2c3
Clock gate name: BI2c3.

enumerator kCLOCK_BI2c4
Clock gate name: BI2c4.

enumerator kCLOCK_BI2c5
Clock gate name: BI2c5.

enumerator kCLOCK_BI2c6
Clock gate name: BI2c6.

enumerator kCLOCK_BI2c7
Clock gate name: BI2c7.

enumerator kCLOCK_FlexI2s0
Clock gate name: FlexI2s0.

enumerator kCLOCK_FlexI2s1
Clock gate name: FlexI2s1.

enumerator kCLOCK_FlexI2s2

 Clock gate name: FlexI2s2.

enumerator kCLOCK_FlexI2s3

 Clock gate name: FlexI2s3.

enumerator kCLOCK_FlexI2s4

 Clock gate name: FlexI2s4.

enumerator kCLOCK_FlexI2s5

 Clock gate name: FlexI2s5.

enumerator kCLOCK_FlexI2s6

 Clock gate name: FlexI2s6.

enumerator kCLOCK_FlexI2s7

 Clock gate name: FlexI2s7.

enumerator kCLOCK_Dmic

 Clock gate name: Dmic.

enumerator kCLOCK_Timer2

 Clock gate name: Timer2.

enumerator kCLOCK_Usbd0

 Clock gate name: Usbd0.

enumerator kCLOCK_Timer0

 Clock gate name: Timer0.

enumerator kCLOCK_Timer1

 Clock gate name: Timer1.

enumerator kCLOCK_Ezhib

 Clock gate name: Ezhib.

enumerator kCLOCK_Dma1

 Clock gate name: Dma1.

enumerator kCLOCK_Comp

 Clock gate name: Comp.

enumerator kCLOCK_Freqme

 Clock gate name: Freqme.

enumerator kCLOCK_Cdog

 Clock gate name: Cdog.

enumerator kCLOCK_Rng

 Clock gate name: Rng.

enumerator kCLOCK_Pmux1

 Clock gate name: Pmux1.

enumerator kCLOCK_Sysctl

 Clock gate name: Sysctl.

enumerator kCLOCK_Usbhmr0

 Clock gate name: Usbhmr0.

enumerator kCLOCK_Usbhs0

 Clock gate name: Usbhs0.

enumerator kCLOCK_Css
Clock gate name: Css.

enumerator kCLOCK_PowerQuad
Clock gate name: PowerQuad.

enumerator kCLOCK_Timer3
Clock gate name: Timer3.

enumerator kCLOCK_Timer4
Clock gate name: Timer4.

enumerator kCLOCK_Puf
Clock gate name: Puf.

enumerator kCLOCK_Pkc
Clock gate name: Pkc.

enumerator kCLOCK_AnalogCtrl
Clock gate name: AnalogCtrl.

enumerator kCLOCK_Hs_Lspi
Clock gate name: Lspi.

enumerator kCLOCK_Gpio_Sec
Clock gate name: Sec.

enumerator kCLOCK_Gpio_Sec_Int
Clock gate name: Int.

enumerator kCLOCK_I3c0
Clock gate name: I3c0.

enumerator kCLOCK_Enc0
Clock gate name: Enc0.

enumerator kCLOCK_Enc1
Clock gate name: Enc1.

enumerator kCLOCK_Pwm0
Clock gate name: Pwm0.

enumerator kCLOCK_Pwm1
Clock gate name: Pwm1.

enumerator kCLOCK_Aoi0
Clock gate name: Aoi0.

enumerator kCLOCK_Aoi1
Clock gate name: Aoi1.

enumerator kCLOCK_Ftm0
Clock gate name: Ftm0.

enumerator kCLOCK_Dac1
Clock gate name: Dac1.

enumerator kCLOCK_Dac2
Clock gate name: Dac2.

enumerator kCLOCK_Opamp0
Clock gate name: Opamp0.

enumerator kCLOCK_Opamp1
Clock gate name: Opamp1.

enumerator kCLOCK_Opamp2
Clock gate name: Opamp2.

enumerator kCLOCK_Hscmp0
Clock gate name: Hscmp0.

enumerator kCLOCK_Hscmp1
Clock gate name: Hscmp1.

enumerator kCLOCK_Hscmp2
Clock gate name: Hscmp2.

enumerator kCLOCK_Vref
Clock gate name: Vref.

enumerator kCLOCK_Pwm0_Sm0
Clock gate name: PWM0 SM0.

enumerator kCLOCK_Pwm0_Sm1
Clock gate name: PWM0 SM1.

enumerator kCLOCK_Pwm0_Sm2
Clock gate name: PWM0 SM2.

enumerator kCLOCK_Pwm0_Sm3
Clock gate name: PWM0 SM3.

enumerator kCLOCK_Pwm1_Sm0
Clock gate name: PWM1 SM0.

enumerator kCLOCK_Pwm1_Sm1
Clock gate name: PWM1 SM1.

enumerator kCLOCK_Pwm1_Sm2
Clock gate name: PWM1 SM2.

enumerator kCLOCK_Pwm1_Sm3
Clock gate name: PWM1 SM3.

enum _clock_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_CoreSysClk
Core/system clock (aka MAIN_CLK)

enumerator kCLOCK_BusClk
Bus clock (AHB clock)

enumerator kCLOCK_ClockOut
CLOCKOUT

enumerator kCLOCK_FroHf
FRO48/96

enumerator kCLOCK_Pll1Out
PLL1 Output

enumerator kCLOCK_Mclk
MCLK

enumerator kCLOCK_Fro12M
FRO12M

enumerator kCLOCK_Fro1M
FRO1M

enumerator kCLOCK_ExtClk
External Clock

enumerator kCLOCK_Pll0Out
PLL0 Output

enumerator kCLOCK_PllClkDiv
PLLCLKDIV clock

enumerator kCLOCK_FlexI2S
FlexI2S clock

enum _clock_attach_id

The enumerator of clock attach Id.

Values:

enumerator kFRO12M_to_MAIN_CLK
Attach FRO12M to MAIN_CLK.

enumerator kEXT_CLK_to_MAIN_CLK
Attach EXT_CLK to MAIN_CLK.

enumerator kFRO1M_to_MAIN_CLK
Attach FRO1M to MAIN_CLK.

enumerator kFRO_HF_to_MAIN_CLK
Attach FRO_HF to MAIN_CLK.

enumerator kPLL0_to_MAIN_CLK
Attach PLL0 to MAIN_CLK.

enumerator kPLL1_to_MAIN_CLK
Attach PLL1 to MAIN_CLK.

enumerator kOSC32K_to_MAIN_CLK
Attach OSC32K to MAIN_CLK.

enumerator kSYSTICK_DIV_to_SYSTICK0
Attach SYSTICK_DIV to SYSTICK0.

enumerator kFRO1M_to_SYSTICK0
Attach FRO1M to SYSTICK0.

enumerator kOSC32K_to_SYSTICK0
Attach OSC32K to SYSTICK0.

enumerator kNONE_to_SYSTICK0
Attach NONE to SYSTICK0.

enumerator kTRACE_DIV_to_TRACE
Attach TRACE_DIV to TRACE.

enumerator kFRO1M_to_TRACE
Attach FRO1M to TRACE.

enumerator kOSC32K_to_TRACE
Attach OSC32K to TRACE.

enumerator kNONE_to_TRACE
Attach NONE to TRACE.

enumerator kMAIN_CLK_to_CTIMER0
Attach MAIN_CLK to CTIMER0.

enumerator kPLL0_to_CTIMER0
Attach PLL0 to CTIMER0.

enumerator kPLL1_to_CTIMER0
Attach PLL1 to CTIMER0.

enumerator kFRO_HF_to_CTIMER0
Attach FRO_HF to CTIMER0.

enumerator kFRO1M_to_CTIMER0
Attach FRO1M to CTIMER0.

enumerator kMCLK_IN_to_CTIMER0
Attach MCLK_IN to CTIMER0.

enumerator kOSC32K_to_CTIMER0
Attach OSC32K to CTIMER0.

enumerator kNONE_to_CTIMER0
Attach NONE to CTIMER0.

enumerator kMAIN_CLK_to_CTIMER1
Attach MAIN_CLK to CTIMER1.

enumerator kPLL0_to_CTIMER1
Attach PLL0 to CTIMER1.

enumerator kPLL1_to_CTIMER1
Attach PLL1 to CTIMER1.

enumerator kFRO_HF_to_CTIMER1
Attach FRO_HF to CTIMER1.

enumerator kFRO1M_to_CTIMER1
Attach FRO1M to CTIMER1.

enumerator kMCLK_IN_to_CTIMER1
Attach MCLK_IN to CTIMER1.

enumerator kOSC32K_to_CTIMER1
Attach OSC32K to CTIMER1.

enumerator kNONE_to_CTIMER1
Attach NONE to CTIMER1.

enumerator kMAIN_CLK_to_CTIMER2
Attach MAIN_CLK to CTIMER2.

enumerator kPLL0_to_CTIMER2
Attach PLL0 to CTIMER2.

enumerator kPLL1_to_CTIMER2
Attach PLL1 to CTIMER2.

enumerator kFRO_HF_to_CTIMER2
Attach FRO_HF to CTIMER2.

enumerator kFRO1M_to_CTIMER2
Attach FRO1M to CTIMER2.

enumerator kMCLK_IN_to_CTIMER2
Attach MCLK_IN to CTIMER2.

enumerator kOSC32K_to_CTIMER2
Attach OSC32K to CTIMER2.

enumerator kNONE_to_CTIMER2
Attach NONE to CTIMER2.

enumerator kMAIN_CLK_to_CTIMER3
Attach MAIN_CLK to CTIMER3.

enumerator kPLL0_to_CTIMER3
Attach PLL0 to CTIMER3.

enumerator kPLL1_to_CTIMER3
Attach PLL1 to CTIMER3.

enumerator kFRO_HF_to_CTIMER3
Attach FRO_HF to CTIMER3.

enumerator kFRO1M_to_CTIMER3
Attach FRO1M to CTIMER3.

enumerator kMCLK_IN_to_CTIMER3
Attach MCLK_IN to CTIMER3.

enumerator kOSC32K_to_CTIMER3
Attach OSC32K to CTIMER3.

enumerator kNONE_to_CTIMER3
Attach NONE to CTIMER3.

enumerator kMAIN_CLK_to_CTIMER4
Attach MAIN_CLK to CTIMER4.

enumerator kPLL0_to_CTIMER4
Attach PLL0 to CTIMER4.

enumerator kPLL1_to_CTIMER4
Attach PLL1 to CTIMER4.

enumerator kFRO_HF_to_CTIMER4
Attach FRO_HF to CTIMER4.

enumerator kFRO1M_to_CTIMER4
Attach FRO1M to CTIMER4.

enumerator kMCLK_IN_to_CTIMER4
Attach MCLK_IN to CTIMER4.

enumerator kOSC32K_to_CTIMER4
Attach OSC32K to CTIMER4.

enumerator kNONE_to_CTIMER4

Attach NONE to CTIMER4.

enumerator kMAIN_CLK_to_CLKOUT

Attach MAIN_CLK to CLKOUT.

enumerator kPLL0_to_CLKOUT

Attach PLL0 to CLKOUT.

enumerator kEXT_CLK_to_CLKOUT

Attach EXT_CLK to CLKOUT.

enumerator kFRO_HF_to_CLKOUT

Attach FRO_HF to CLKOUT.

enumerator kFRO1M_to_CLKOUT

Attach FRO1M to CLKOUT.

enumerator kPLL1_to_CLKOUT

Attach PLL1 to CLKOUT.

enumerator kOSC32K_to_CLKOUT

Attach OSC32K to CLKOUT.

enumerator kNONE_to_CLKOUT

Attach NONE to CLKOUT.

enumerator kFRO12M_to_PLL0

Attach FRO12M to PLL0.

enumerator kEXT_CLK_to_PLL0

Attach EXT_CLK to PLL0.

enumerator kFRO1M_to_PLL0

Attach FRO1M to PLL0.

enumerator kOSC32K_to_PLL0

Attach OSC32K to PLL0.

enumerator kNONE_to_PLL0

Attach NONE to PLL0.

enumerator kFRO12M_to_PLL1

Attach FRO12M to PLL1.

enumerator kEXT_CLK_to_PLL1

Attach EXT_CLK to PLL1.

enumerator kFRO1M_to_PLL1

Attach FRO1M to PLL1.

enumerator kOSC32K_to_PLL1

Attach OSC32K to PLL1.

enumerator kNONE_to_PLL1

Attach NONE to PLL1.

enumerator kMCAN_DIV_to_MCAN

Attach MCAN_DIV to MCAN.

enumerator kFRO1M_to_MCAN

Attach FRO1M to MCAN.

enumerator kOSC32K_to_MCAN
Attach OSC32K to MCAN.

enumerator kNONE_to_MCAN
Attach NONE to MCAN.

enumerator kMAIN_CLK_to_ADC0
Attach MAIN_CLK to ADC0.

enumerator kPLL0_to_ADC0
Attach PLL0 to ADC0.

enumerator kFRO_HF_to_ADC0
Attach FRO_HF to ADC0.

enumerator kEXT_CLK_to_ADC0
Attach XO to ADC0.

enumerator kNONE_to_ADC0
Attach NONE to ADC0.

enumerator kMAIN_CLK_to_ADC1
Attach MAIN_CLK to ADC1.

enumerator kPLL0_to_ADC1
Attach PLL0 to ADC1.

enumerator kFRO_HF_to_ADC1
Attach FRO_HF to ADC1.

enumerator kEXT_CLK_to_ADC1
Attach XO to ADC1.

enumerator kNONE_to_ADC1
Attach NONE to ADC1.

enumerator kMAIN_CLK_to_USB0
Attach MAIN_CLK to USB0.

enumerator kPLL0_to_USB0
Attach PLL0 to USB0.

enumerator kFRO_HF_to_USB0
Attach FRO_HF to USB0.

enumerator kPLL1_to_USB0
Attach PLL1 to USB0.

enumerator kNONE_to_USB0
Attach NONE to USB0.

enumerator kMAIN_CLK_to_FLEXCOMM0
Attach MAIN_CLK to FLEXCOMM0.

enumerator kMAIN_CLK_FRG0_to_FLEXCOMM0
Attach Main clock to FlexComm0.

enumerator kPLL_CLK_DIV_FRG0_to_FLEXCOMM0
Attach PLL clock DIV Frg to FlexComm0.

enumerator kFRO_HF_DIV_FRG0_to_FLEXCOMM0
Attach FRO HF DIV FRG to FlexComm0.

enumerator kFRO12M_to_FLEXCOMM0
Attach FRO12M to FLEXCOMM0.

enumerator kFRO_HF_DIV_to_FLEXCOMM0
Attach FRO_HF_DIV to FLEXCOMM0.

enumerator kFRO1M_to_FLEXCOMM0
Attach FRO1M to FLEXCOMM0.

enumerator kMCLK_IN_to_FLEXCOMM0
Attach MCLK_IN to FLEXCOMM0.

enumerator kOSC32K_to_FLEXCOMM0
Attach OSC32K to FLEXCOMM0.

enumerator kNONE_to_FLEXCOMM0
Attach NONE to FLEXCOMM0.

enumerator kMAIN_CLK_to_FLEXCOMM1
Attach MAIN_CLK to FLEXCOMM1.

enumerator kMAIN_CLK_FRG1_to_FLEXCOMM1
Attach Main clock to FlexComm1.

enumerator kPLL_CLK_DIV_FRG1_to_FLEXCOMM1
Attach PLL clock DIV Frg to FlexComm1.

enumerator kFRO_HF_DIV_FRG1_to_FLEXCOMM1
Attach FRO HF DIV FRG to FlexComm1.

enumerator kFRO12M_to_FLEXCOMM1
Attach FRO12M to FLEXCOMM1.

enumerator kFRO_HF_DIV_to_FLEXCOMM1
Attach FRO_HF_DIV to FLEXCOMM1.

enumerator kFRO1M_to_FLEXCOMM1
Attach FRO1M to FLEXCOMM1.

enumerator kMCLK_IN_to_FLEXCOMM1
Attach MCLK_IN to FLEXCOMM1.

enumerator kOSC32K_to_FLEXCOMM1
Attach OSC32K to FLEXCOMM1.

enumerator kNONE_to_FLEXCOMM1
Attach NONE to FLEXCOMM1.

enumerator kMAIN_CLK_to_FLEXCOMM2
Attach MAIN_CLK to FLEXCOMM2.

enumerator kMAIN_CLK_FRG2_to_FLEXCOMM2
Attach Main clock to FlexComm2.

enumerator kPLL_CLK_DIV_FRG2_to_FLEXCOMM2
Attach PLL clock DIV Frg to FlexComm2.

enumerator kFRO_HF_DIV_FRG2_to_FLEXCOMM2
Attach FRO HF DIV FRG to FlexComm2.

enumerator kFRO12M_to_FLEXCOMM2
Attach FRO12M to FLEXCOMM2.

enumerator kFRO_HF_DIV_to_FLEXCOMM2
Attach FRO_HF_DIV to FLEXCOMM2.

enumerator kFRO1M_to_FLEXCOMM2
Attach FRO1M to FLEXCOMM2.

enumerator kMCLK_IN_to_FLEXCOMM2
Attach MCLK_IN to FLEXCOMM2.

enumerator kOSC32K_to_FLEXCOMM2
Attach OSC32K to FLEXCOMM2.

enumerator kNONE_to_FLEXCOMM2
Attach NONE to FLEXCOMM2.

enumerator kMAIN_CLK_to_FLEXCOMM3
Attach MAIN_CLK to FLEXCOMM3.

enumerator kMAIN_CLK_FRG3_to_FLEXCOMM3
Attach Main clock to FlexComm3.

enumerator kPLL_CLK_DIV_FRG3_to_FLEXCOMM3
Attach PLL clock DIV Frg to FlexComm3.

enumerator kFRO_HF_DIV_FRG3_to_FLEXCOMM3
Attach FRO HF DIV FRG to FlexComm3.

enumerator kFRO12M_to_FLEXCOMM3
Attach FRO12M to FLEXCOMM3.

enumerator kFRO_HF_DIV_to_FLEXCOMM3
Attach FRO_HF_DIV to FLEXCOMM3.

enumerator kFRO1M_to_FLEXCOMM3
Attach FRO1M to FLEXCOMM3.

enumerator kMCLK_IN_to_FLEXCOMM3
Attach MCLK_IN to FLEXCOMM3.

enumerator kOSC32K_to_FLEXCOMM3
Attach OSC32K to FLEXCOMM3.

enumerator kNONE_to_FLEXCOMM3
Attach NONE to FLEXCOMM3.

enumerator kMAIN_CLK_to_FLEXCOMM4
Attach MAIN_CLK to FLEXCOMM4.

enumerator kMAIN_CLK_FRG4_to_FLEXCOMM4
Attach Main clock to FlexComm4.

enumerator kPLL_CLK_DIV_FRG4_to_FLEXCOMM4
Attach PLL clock DIV Frg to FlexComm4.

enumerator kFRO_HF_DIV_FRG4_to_FLEXCOMM4
Attach FRO HF DIV FRG to FlexComm4.

enumerator kFRO12M_to_FLEXCOMM4
Attach FRO12M to FLEXCOMM4.

enumerator kFRO_HF_DIV_to_FLEXCOMM4
Attach FRO_HF_DIV to FLEXCOMM4.

enumerator kFRO1M_to_FLEXCOMM4
Attach FRO1M to FLEXCOMM4.

enumerator kMCLK_IN_to_FLEXCOMM4
Attach MCLK_IN to FLEXCOMM4.

enumerator kOSC32K_to_FLEXCOMM4
Attach OSC32K to FLEXCOMM4.

enumerator kNONE_to_FLEXCOMM4
Attach NONE to FLEXCOMM4.

enumerator kMAIN_CLK_to_FLEXCOMM5
Attach MAIN_CLK to FLEXCOMM5.

enumerator kMAIN_CLK_FRG5_to_FLEXCOMM5
Attach Main clock to FlexComm5.

enumerator kPLL_CLK_DIV_FRG5_to_FLEXCOMM5
Attach PLL clock DIV Frg to FlexComm5.

enumerator kFRO_HF_DIV_FRG5_to_FLEXCOMM5
Attach FRO HF DIV FRG to FlexComm5.

enumerator kFRO12M_to_FLEXCOMM5
Attach FRO12M to FLEXCOMM5.

enumerator kFRO_HF_DIV_to_FLEXCOMM5
Attach FRO_HF_DIV to FLEXCOMM5.

enumerator kFRO1M_to_FLEXCOMM5
Attach FRO1M to FLEXCOMM5.

enumerator kMCLK_IN_to_FLEXCOMM5
Attach MCLK_IN to FLEXCOMM5.

enumerator kOSC32K_to_FLEXCOMM5
Attach OSC32K to FLEXCOMM5.

enumerator kNONE_to_FLEXCOMM5
Attach NONE to FLEXCOMM5.

enumerator kMAIN_CLK_to_FLEXCOMM6
Attach MAIN_CLK to FLEXCOMM6.

enumerator kMAIN_CLK_FRG6_to_FLEXCOMM6
Attach Main clock to FlexComm6.

enumerator kPLL_CLK_DIV_FRG6_to_FLEXCOMM6
Attach PLL clock DIV Frg to FlexComm6.

enumerator kFRO_HF_DIV_FRG6_to_FLEXCOMM6
Attach FRO HF DIV FRG to FlexComm6.

enumerator kFRO12M_to_FLEXCOMM6
Attach FRO12M to FLEXCOMM6.

enumerator kFRO_HF_DIV_to_FLEXCOMM6
Attach FRO_HF_DIV to FLEXCOMM6.

enumerator kFRO1M_to_FLEXCOMM6
Attach FRO1M to FLEXCOMM6.

enumerator kMCLK_IN_to_FLEXCOMM6
Attach MCLK_IN to FLEXCOMM6.

enumerator kOSC32K_to_FLEXCOMM6
Attach OSC32K to FLEXCOMM6.

enumerator kNONE_to_FLEXCOMM6
Attach NONE to FLEXCOMM6.

enumerator kMAIN_CLK_to_FLEXCOMM7
Attach MAIN_CLK to FLEXCOMM7.

enumerator kMAIN_CLK_FRG7_to_FLEXCOMM7
Attach Main clock to FlexComm7.

enumerator kPLL_CLK_DIV_FRG7_to_FLEXCOMM7
Attach PLL clock DIV Frg to FlexComm7.

enumerator kFRO_HF_DIV_FRG7_to_FLEXCOMM7
Attach PLL clock DIV Frg to FlexComm7.

enumerator kFRO12M_to_FLEXCOMM7
Attach FRO12M to FLEXCOMM7.

enumerator kFRO_HF_DIV_to_FLEXCOMM7
Attach FRO_HF_DIV to FLEXCOMM7.

enumerator kFRO1M_to_FLEXCOMM7
Attach FRO1M to FLEXCOMM7.

enumerator kMCLK_IN_to_FLEXCOMM7
Attach MCLK_IN to FLEXCOMM7.

enumerator kOSC32K_to_FLEXCOMM7
Attach OSC32K to FLEXCOMM7.

enumerator kNONE_to_FLEXCOMM7
Attach NONE to FLEXCOMM7.

enumerator kMAIN_CLK_to_HSLSPI
Attach MAIN_CLK to HSLSPI.

enumerator kPLL_CLK_DIV_to_HSLSPI
Attach PLL_CLK_DIV to HSLSPI.

enumerator kFRO12M_to_HSLSPI
Attach FRO12M to HSLSPI.

enumerator kFRO_HF_DIV_to_HSLSPI
Attach FRO_HF_DIV to HSLSPI.

enumerator kFRO1M_to_HSLSPI
Attach FRO1M to HSLSPI.

enumerator kOSC32K_to_HSLSPI
Attach OSC32K to HSLSPI.

enumerator kNONE_to_HSLSPI
Attach NONE to HSLSPI.

enumerator kFRO_HF_to_MCLK
Attach FRO_HF to MCLK.

enumerator kPLL0_to_MCLK
Attach PLL0 to MCLK.

enumerator kNONE_to_MCLK
Attach NONE to MCLK.

enumerator kMAIN_CLK_to_SCT
Attach MAIN_CLK to SCT.

enumerator kPLL0_to_SCT
Attach PLL0 to SCT.

enumerator kEXT_CLK_to_SCT
Attach EXT_CLK to SCT.

enumerator kFRO_HF_to_SCT
Attach FRO_HF to SCT.

enumerator kPLL1_to_SCT
Attach PLL1 to SCT.

enumerator kMCLK_IN_to_SCT
Attach MCLK_IN to SCT.

enumerator kNONE_to_SCT
Attach NONE to SCT.

enumerator kMAIN_CLK_to_DAC0
Attach MAIN_CLK to DAC0.

enumerator kPLL0_to_DAC0
Attach PLL0 to DAC0.

enumerator kFRO_HF_to_DAC0
Attach FRO_HF to DAC0.

enumerator kFRO12M_to_DAC0
Attach FRO12M to DAC0.

enumerator kPLL1_to_DAC0
Attach PLL1 to DAC0.

enumerator kFRO1M_to_DAC0
Attach FRO1M to DAC0.

enumerator kNONE_to_DAC0
Attach NONE to DAC0.

enumerator kMAIN_CLK_to_DAC1
Attach MAIN_CLK to DAC1.

enumerator kPLL0_to_DAC1
Attach PLL0 to DAC1.

enumerator kFRO_HF_to_DAC1
Attach FRO_HF to DAC1.

enumerator kFRO12M_to_DAC1
Attach FRO12M to DAC1.

enumerator kPLL1_to_DAC1
Attach PLL1 to DAC1.

enumerator kFRO1M_to_DAC1
Attach FRO1M to DAC1.

enumerator kNONE_to_DAC1
Attach NONE to DAC1.

enumerator kMAIN_CLK_to_DAC2
Attach MAIN_CLK to DAC2.

enumerator kPLL0_to_DAC2
Attach PLL0 to DAC2.

enumerator kFRO_HF_to_DAC2
Attach FRO_HF to DAC2.

enumerator kFRO12M_to_DAC2
Attach FRO12M to DAC2.

enumerator kPLL1_to_DAC2
Attach PLL1 to DAC2.

enumerator kFRO1M_to_DAC2
Attach FRO1M to DAC2.

enumerator kNONE_to_DAC2
Attach NONE to DAC2.

enumerator kMAIN_CLK_to_FLEXSPI
Attach MAIN_CLK to FLEXSPI.

enumerator kPLL0_to_FLEXSPI
Attach PLL0 to FLEXSPI.

enumerator kFRO_HF_to_FLEXSPI
Attach FRO_HF to FLEXSPI.

enumerator kPLL1_to_FLEXSPI
Attach PLL1 to FLEXSPI.

enumerator kNONE_to_FLEXSPI
Attach NONE to FLEXSPI.

enumerator kPLL0_to_PLLCLKDIV
Attach PLL0 to PLLCLKDIV.

enumerator kPLL1_to_PLLCLKDIV
Attach PLL1 to PLLCLKDIV.

enumerator kNONE_to_PLLCLKDIV
Attach NONE to PLLCLKDIV.

enumerator kMAIN_CLK_to_I3CFCLK
Attach MAIN_CLK to I3CFCLK.

enumerator kFRO_HF_DIV_to_I3CFCLK
Attach FRO_HF_DIV to I3CFCLK.

enumerator kNONE_to_I3CFCLK
Attach NONE to I3CFCLK.

enumerator kI3CFCLKSEL_to_I3CFCLKSTC
Attach I3CFCLKSEL to I3CFCLKSTC.

enumerator kFRO1M_to_I3CFCLKSTC

Attach FRO1M to I3CFCLKSTC.

enumerator kNONE_to_I3CFCLKSTC

Attach NONE to I3CFCLKSTC.

enumerator kMAIN_CLK_to_DMIC

Attach MAIN_CLK to DMIC.

enumerator kPLL0_to_DMIC

Attach PLL0 to DMIC.

enumerator kEXT_CLK_to_DMIC

Attach EXT_CLK to DMIC.

enumerator kFRO_HF_to_DMIC

Attach FRO_HF to DMIC.

enumerator kPLL1_to_DMIC

Attach PLL1 to DMIC.

enumerator kMCLK_IN_to_DMIC

Attach MCLK_IN to DMIC.

enumerator kNONE_to_DMIC

Attach NONE to DMIC.

enumerator kFRO32K_to_FCOSC32K

Attach FRO32K to FCOSC32K.

enumerator kXTAL32K_to_FCOSC32K

Attach XTAL32K to FCOSC32K.

enumerator kFRO32K_to_OSC32K

Attach FRO32K to OSC32K.

enumerator kXTAL32K_to_OSC32K

Attach XTAL32K to OSC32K.

enumerator kFRO32K_to_FC32K

Attach FRO32K to FC32K.

enumerator kXTAL32K_to_FC32K

Attach XTAL32K to FC32K.

enumerator kFRO32K_to_OSTIMER

Attach FRO32K to OSTIMER.

enumerator kOSC32K_to_OSTIMER

Attach OSC32K to OSTIMER.

enumerator kFRO1M_to_OSTIMER

Attach FRO1M to OSTIMER.

enumerator kAHB_CLK_to_OSTIMER

Attach AHB_CLK to OSTIMER.

enumerator kNONE_to_NONE

Attach NONE to NONE.

enum _clock_div_name

Clock dividers.

Values:

enumerator kCLOCK_DivSystickClk
Systick Clock Divider.

enumerator kCLOCK_DivArmTrClkDiv
Trace Clock Divider.

enumerator kCLOCK_DivCanClk
Can Clock Divider.

enumerator kCLOCK_DivFlexFrg0
FRGCTRL0 register.

enumerator kCLOCK_DivFlexFrg1
FRGCTRL1 register.

enumerator kCLOCK_DivFlexFrg2
FRGCTRL2 register.

enumerator kCLOCK_DivFlexFrg3
FRGCTRL3 register.

enumerator kCLOCK_DivFlexFrg4
FRGCTRL4 register.

enumerator kCLOCK_DivFlexFrg5
FRGCTRL5 register.

enumerator kCLOCK_DivFlexFrg6
FRGCTRL6 register.

enumerator kCLOCK_DivFlexFrg7
FRGCTRL7 register.

enumerator kCLOCK_DivAhbClk
Ahb Clock Divider.

enumerator kCLOCK_DivClkOut
Clk Out Divider.

enumerator kCLOCK_DivFrohfClk
Frohf Divider.

enumerator kCLOCK_DivWdtClk
Wdt Clock Divider.

enumerator kCLOCK_DivAdc0Clk
Adc0 Clock Divider.

enumerator kCLOCK_DivUsb0Clk
Usb0 Clock Divider.

enumerator kCLOCK_DivMclk
Mclk Divider.

enumerator kCLOCK_DivSctClk
Sct Clock Divider.

enumerator kCLOCK_DivPllClk
Pll0 Clock Divider.

enumerator kCLOCK_DivCtimer0Clk
Ctimer0 Clock Divider.

enumerator kCLOCK_DivCtimer1Clk
Ctimer1 Clock Divider.

enumerator kCLOCK_DivCtimer2Clk
Ctimer2 Clock Divider.

enumerator kCLOCK_DivCtimer3Clk
Ctimer3 Clock Divider.

enumerator kCLOCK_DivCtimer4Clk
Ctimer4 Clock Divider.

enumerator kCLOCK_DivAdc1Clk
Adc1 Clock Divider.

enumerator kCLOCK_DivDac0Clk
Dac0 Clock Divider.

enumerator kCLOCK_DivDac1Clk
Dac1 Clock Divider.

enumerator kCLOCK_DivDac2Clk
Dac2 Clock Divider.

enumerator kCLOCK_DivFlexSpiClk
Flex Spi Clock Divider.

enumerator kCLOCK_DivI3cFclkStc
I3c Fclk Stc Divider.

enumerator kCLOCK_DivI3cFclkS
I3c Fclk S Divider.

enumerator kCLOCK_DivI3cFclk
I3c Fclk Divider.

enumerator kCLOCK_DivDmicClk
Dmic Clock Divider.

enumerator kCLOCK_DivFlexcom0Clk
Flexcom0 Clock Divider.

enumerator kCLOCK_DivFlexcom1Clk
Flexcom1 Clock Divider.

enumerator kCLOCK_DivFlexcom2Clk
Flexcom2 Clock Divider.

enumerator kCLOCK_DivFlexcom3Clk
Flexcom3 Clock Divider.

enumerator kCLOCK_DivFlexcom4Clk
Flexcom4 Clock Divider.

enumerator kCLOCK_DivFlexcom5Clk
Flexcom5 Clock Divider.

enumerator kCLOCK_DivFlexcom6Clk
Flexcom6 Clock Divider.

enumerator kCLOCK_DivFlexcom7Clk
Flexcom7 Clock Divider.

enum _ss_progmodfm

PLL Spread Spectrum (SS) Programmable modulation frequency See (MF) field in the PLL0SSCG1 register in the UM.

Values:

enumerator kSS_MF_512

Nss = 512 (fm ? 3.9 - 7.8 kHz)

enumerator kSS_MF_384

Nss ?= 384 (fm ? 5.2 - 10.4 kHz)

enumerator kSS_MF_256

Nss = 256 (fm ? 7.8 - 15.6 kHz)

enumerator kSS_MF_128

Nss = 128 (fm ? 15.6 - 31.3 kHz)

enumerator kSS_MF_64

Nss = 64 (fm ? 32.3 - 64.5 kHz)

enumerator kSS_MF_32

Nss = 32 (fm ? 62.5- 125 kHz)

enumerator kSS_MF_24

Nss ?= 24 (fm ? 83.3- 166.6 kHz)

enumerator kSS_MF_16

Nss = 16 (fm ? 125- 250 kHz)

enum _ss_progmoddp

PLL Spread Spectrum (SS) Programmable frequency modulation depth See (MR) field in the PLL0SSCG1 register in the UM.

Values:

enumerator kSS_MR_K0

k = 0 (no spread spectrum)

enumerator kSS_MR_K1

k = 1

enumerator kSS_MR_K1_5

k = 1.5

enumerator kSS_MR_K2

k = 2

enumerator kSS_MR_K3

k = 3

enumerator kSS_MR_K4

k = 4

enumerator kSS_MR_K6

k = 6

enumerator kSS_MR_K8

k = 8

enum _ss_modwvctrl

PLL Spread Spectrum (SS) Modulation waveform control See (MC) field in the PLL0SSCG1 register in the UM.

Compensation for low pass filtering of the PLL to get a triangular modulation at the output of the PLL, giving a flat frequency spectrum.

Values:

enumerator kSS_MC_NOC
no compensation

enumerator kSS_MC_RECC
recommended setting

enumerator kSS_MC_MAXC
max. compensation

enum _pll_error

PLL status definitions.

Values:

enumerator kStatus_PLL_Success
PLL operation was successful

enumerator kStatus_PLL_OutputTooLow
PLL output rate request was too low

enumerator kStatus_PLL_OutputTooHigh
PLL output rate request was too high

enumerator kStatus_PLL_InputTooLow
PLL input rate is too low

enumerator kStatus_PLL_InputTooHigh
PLL input rate is too high

enumerator kStatus_PLL_OutsideIntLimit
Requested output rate isn't possible

enumerator kStatus_PLL_CCOTooLow
Requested CCO rate isn't possible

enumerator kStatus_PLL_CCOTooHigh
Requested CCO rate isn't possible

enum _clock_usbfs_src

USB FS clock source definition.

Values:

enumerator kCLOCK_UsbfsSrcFro
Use FRO 96 MHz.

enumerator kCLOCK_UsbfsSrcPll0
Use PLL0 output.

enumerator kCLOCK_UsbfsSrcMainClock
Use Main clock.

enumerator kCLOCK_UsbfsSrcPll1
Use PLL1 clock.

enumerator kCLOCK__UsbfsSrcNone

this may be selected in order to reduce power when no output is needed.

enum __clock_usb_phy_src

Source of the USB HS PHY.

Values:

enumerator kCLOCK__UsbPhySrcExt

Use external crystal.

typedef enum __clock_ip_name clock_ip_name_t

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

typedef enum __clock_name clock_name_t

Clock name used to get clock frequency.

typedef enum __clock_attach_id clock_attach_id_t

The enumerator of clock attach Id.

typedef enum __clock_div_name clock_div_name_t

Clock dividers.

typedef enum __ss_progmodfm ss_progmodfm_t

PLL Spread Spectrum (SS) Programmable modulation frequency See (MF) field in the PLL0SSCG1 register in the UM.

typedef enum __ss_progmoddp ss_progmoddp_t

PLL Spread Spectrum (SS) Programmable frequency modulation depth See (MR) field in the PLL0SSCG1 register in the UM.

typedef enum __ss_modwvctrl ss_modwvctrl_t

PLL Spread Spectrum (SS) Modulation waveform control See (MC) field in the PLL0SSCG1 register in the UM.

Compensation for low pass filtering of the PLL to get a triangular modulation at the output of the PLL, giving a flat frequency spectrum.

typedef struct __pll_config pll_config_t

PLL configuration structure.

This structure can be used to configure the settings for a PLL setup structure. Fill in the desired configuration for the PLL and call the PLL setup function to fill in a PLL setup structure.

typedef struct __pll_setup pll_setup_t

PLL0 setup structure This structure can be used to pre-build a PLL setup configuration at run-time and quickly set the PLL to the configuration. It can be populated with the PLL setup function. If powering up or waiting for PLL lock, the PLL input clock source should be configured prior to PLL setup.

typedef enum __pll_error pll_error_t

PLL status definitions.

typedef enum __clock_usbfs_src clock_usbfs_src_t

USB FS clock source definition.

typedef enum __clock_usb_phy_src clock_usb_phy_src_t

Source of the USB HS PHY.

static inline void CLOCK_EnableClock(*clock_ip_name_t* clk)

Enable the clock for specific IP.

Parameters

- clk – : Clock to be enabled.

Returns

Nothing

static inline void CLOCK_DisableClock(*clock_ip_name_t* clk)

Disable the clock for specific IP.

Parameters

- clk – : Clock to be Disabled.

Returns

Nothing

status_t CLOCK_SetupFROClocking(uint32_t iFreq)

Initialize the Core clock to given frequency (12, 48 or 96 MHz). Turns on FRO and uses default CCO, if freq is 12000000, then high speed output is off, else high speed output is enabled.

Parameters

- iFreq – : Desired frequency (must be one of CLK_FRO_12MHZ or CLK_FRO_48MHZ or CLK_FRO_96MHZ)

Returns

returns success or fail status.

void CLOCK_SetFLASHAccessCyclesForFreq(uint32_t system_freq_hz)

Set the flash wait states for the input frequency.

Parameters

- system_freq_hz – Input frequency

Returns

Nothing

status_t CLOCK_SetupExtClocking(uint32_t iFreq)

Initialize the external osc clock to given frequency.

Parameters

- iFreq – : Desired frequency (must be equal to exact rate in Hz)

Returns

returns success or fail status.

status_t CLOCK_SetupI2SMClkClocking(uint32_t iFreq)

Initialize the I2S MCLK clock to given frequency.

Parameters

- iFreq – : Desired frequency (must be equal to exact rate in Hz)

Returns

returns success or fail status.

void CLOCK_AttachClk(*clock_attach_id_t* connection)

Configure the clock selection muxes.

Parameters

- connection – : Clock to be configured.

Returns

Nothing

clock_attach_id_t CLOCK_GetClockAttachId(*clock_attach_id_t* attachId)

Get the actual clock attach id. This function uses the offset in input attach id, then it reads the actual source value in the register and combine the offset to obtain an actual attach id.

Parameters

- attachId – : Clock attach id to get.

Returns

Clock source value.

void CLOCK_SetClkDiv(*clock_div_name_t* div_name, uint32_t divided_by_value, bool reset)

Setup peripheral clock dividers.

Parameters

- div_name – : Clock divider name
- divided_by_value – Value to be divided
- reset – : Whether to reset the divider counter.

Returns

Nothing

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Return Frequency of selected clock.

Returns

Frequency of selected clock

uint32_t CLOCK_GetFro12MFreq(void)

Return Frequency of FRO 12MHz.

Returns

Frequency of FRO 12MHz

uint32_t CLOCK_GetFro1MFreq(void)

Return Frequency of FRO 1MHz.

Returns

Frequency of FRO 1MHz

uint32_t CLOCK_GetClockOutClkFreq(void)

Return Frequency of ClockOut.

Returns

Frequency of ClockOut

uint32_t CLOCK_GetMCanClkFreq(void)

Return Frequency of Can Clock.

Returns

Frequency of Can.

uint32_t CLOCK_GetAdcClkFreq(uint32_t id)

Return Frequency of Adc Clock.

Returns

Frequency of Adc.

uint32_t CLOCK_GetUsb0ClkFreq(void)

Return Frequency of Usb0 Clock.

Returns

Frequency of Usb0 Clock.

uint32_t CLOCK__GetMclClkFreq(void)

Return Frequency of MClk Clock.

Returns

Frequency of MClk Clock.

uint32_t CLOCK__GetSctClkFreq(void)

Return Frequency of SCTimer Clock.

Returns

Frequency of SCTimer Clock.

uint32_t CLOCK__GetExtClkFreq(void)

Return Frequency of External Clock.

Returns

Frequency of External Clock. If no external clock is used returns 0.

uint32_t CLOCK__GetWdtClkFreq(void)

Return Frequency of Watchdog.

Returns

Frequency of Watchdog

uint32_t CLOCK__GetFroHfFreq(void)

Return Frequency of High-Freq output of FRO.

Returns

Frequency of High-Freq output of FRO

uint32_t CLOCK__GetPll0OutFreq(void)

Return Frequency of PLL.

Returns

Frequency of PLL

uint32_t CLOCK__GetPll1OutFreq(void)

Return Frequency of USB PLL.

Returns

Frequency of PLL

uint32_t CLOCK__GetPllClkDivFreq(void)

Return Frequency of PLL_CLK_DIV.

Returns

Frequency of PLL_CLK_DIV

uint32_t CLOCK__GetOsc32KFreq(void)

Return Frequency of 32kHz osc.

Returns

Frequency of 32kHz osc

uint32_t CLOCK__GetFC32KFreq(void)

Return Frequency of Flexcomm 32kHz osc.

Returns

Frequency of Flexcomm 32kHz osc

uint32_t CLOCK_GetCoreSysClkFreq(void)

Return Frequency of Core System.

Returns

Frequency of Core System

uint32_t CLOCK_GetI2SMClkFreq(void)

Return Frequency of I2S MCLK Clock.

Returns

Frequency of I2S MCLK Clock

uint32_t CLOCK_GetFrgFreq(uint32_t id)

Return Frequency of FRG Clock.

Returns

Frequency of FRG Clock

uint32_t CLOCK_GetFlexCommClkFreq(uint32_t id)

Return Frequency of FlexComm Clock.

Returns

Frequency of FlexComm Clock

uint32_t CLOCK_GetHsLspiClkFreq(void)

Return Frequency of High speed SPI Clock.

Returns

Frequency of High speed SPI Clock

uint32_t CLOCK_GetCTimerClkFreq(uint32_t id)

Return Frequency of CTimer functional Clock.

Returns

Frequency of CTimer functional Clock

uint32_t CLOCK_GetSystickClkFreq(void)

Return Frequency of SystickClock.

Returns

Frequency of Systick Clock

uint32_t CLOCK_GetFlexSpiClkFreq(void)

Return Frequency of FlexSPI.

Returns

Frequency of FlexSPI Clock

uint32_t CLOCK_GetDmicClkFreq(void)

Return Frequency of DMIC.

Returns

Frequency of DMIC Clock

uint32_t CLOCK_GetDacClkFreq(uint32_t id)

Return Frequency of DAC Clock.

Returns

Frequency of DAC Clock

uint32_t CLOCK_GetI3cSTCClkFreq(void)

Return Frequency of I3C function slow TC Clock.

Returns

Frequency of I3C function slow TC Clock

uint32_t CLOCK_GetI3cSClkFreq(void)

Return Frequency of I3C function slow Clock.

Returns

Frequency of I3C function slow Clock

uint32_t CLOCK_GetI3cClkFreq(void)

Return Frequency of I3C function Clock.

Returns

Frequency of I3C function Clock

uint32_t CLOCK_GetPLL0InClockRate(void)

Return PLL0 input clock rate.

Returns

PLL0 input clock rate

uint32_t CLOCK_GetPLL1InClockRate(void)

Return PLL1 input clock rate.

Returns

PLL1 input clock rate

uint32_t CLOCK_GetPLL0OutClockRate(bool recompute)

Return PLL0 output clock rate.

Note: The PLL rate is cached in the driver in a variable as the rate computation function can take some time to perform. It is recommended to use 'false' with the 'recompute' parameter.

Parameters

- recompute – : Forces a PLL rate recomputation if true

Returns

PLL0 output clock rate

uint32_t CLOCK_GetPLL1OutClockRate(bool recompute)

Return PLL1 output clock rate.

Note: The PLL rate is cached in the driver in a variable as the rate computation function can take some time to perform. It is recommended to use 'false' with the 'recompute' parameter.

Parameters

- recompute – : Forces a PLL rate recomputation if true

Returns

PLL1 output clock rate

__STATIC_INLINE void CLOCK_SetBypassPLL0 (bool bypass)

Enables and disables PLL0 bypass mode.

bypass : true to bypass PLL0 (PLL0 output = PLL0 input, false to disable bypass)

Returns

PLL0 output clock rate

__STATIC_INLINE void CLOCK_SetBypassPLL1 (bool bypass)

Enables and disables PLL1 bypass mode.

bypass : true to bypass PLL1 (PLL1 output = PLL1 input, false to disable bypass)

Returns

PLL1 output clock rate

__STATIC_INLINE bool CLOCK_IsPLL0Locked (void)

Check if PLL is locked or not.

Returns

true if the PLL is locked, false if not locked

__STATIC_INLINE bool CLOCK_IsPLL1Locked (void)

Check if PLL1 is locked or not.

Returns

true if the PLL1 is locked, false if not locked

void CLOCK_SetStoredPLL0ClockRate(uint32_t rate)

Store the current PLL0 rate.

Parameters

- rate – Current rate of the PLL0

Returns

Nothing

uint32_t CLOCK_GetPLL0OutFromSetup(*pll_setup_t* *pSetup)

Return PLL0 output clock rate from setup structure.

Parameters

- pSetup – : Pointer to a PLL setup structure

Returns

System PLL output clock rate the setup structure will generate

uint32_t CLOCK_GetPLL1OutFromSetup(*pll_setup_t* *pSetup)

Return PLL1 output clock rate from setup structure.

Parameters

- pSetup – : Pointer to a PLL setup structure

Returns

PLL0 output clock rate the setup structure will generate

pll_error_t CLOCK_SetupPLL0Data(*pll_config_t* *pControl, *pll_setup_t* *pSetup)

Set PLL0 output based on the passed PLL setup data.

Note: Actual frequency for setup may vary from the desired frequency based on the accuracy of input clocks, rounding, non-fractional PLL mode, etc.

Parameters

- pControl – : Pointer to populated PLL control structure to generate setup with
- pSetup – : Pointer to PLL setup structure to be filled

Returns

PLL_ERROR_SUCCESS on success, or PLL setup error code

pll_error_t CLOCK_SetupPLL0Prec(*pll_setup_t* *pSetup, uint32_t flagcfg)

Set PLL output from PLL setup structure (precise frequency)

Note: This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

Parameters

- pSetup – : Pointer to populated PLL setup structure
- flagcfg – : Flag configuration for PLL config structure

Returns

PLL_ERROR_SUCCESS on success, or PLL setup error code

pll_error_t CLOCK_SetPLL0Freq(const *pll_setup_t* *pSetup)

Set PLL output from PLL setup structure (precise frequency)

Note: This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

Parameters

- pSetup – : Pointer to populated PLL setup structure

Returns

kStatus_PLL_Success on success, or PLL setup error code

pll_error_t CLOCK_SetPLL1Freq(const *pll_setup_t* *pSetup)

Set PLL output from PLL setup structure (precise frequency)

Note: This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

Parameters

- pSetup – : Pointer to populated PLL setup structure

Returns

kStatus_PLL_Success on success, or PLL setup error code

void CLOCK_SetupPLL0Mult(uint32_t multiply_by, uint32_t input_freq)

Set PLL0 output based on the multiplier and input frequency.

Note: Unlike the Chip_Clock_SetupSystemPLLPrec() function, this function does not disable or enable PLL power, wait for PLL lock, or adjust system voltages. These must be done in the application. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

Parameters

- multiply_by – : multiplier
- input_freq – : Clock input frequency of the PLL

Returns

Nothing

static inline void CLOCK_DisableUsbDevicefs0Clock(*clock_ip_name_t* clk)

Disable USB clock.

Disable USB clock.

bool CLOCK_EnableUsbfs0DeviceClock(*clock_usbfs_src_t* src, uint32_t freq)

Enable USB Device FS clock.

Parameters

- src – : clock source
- freq – clock frequency Enable USB Device Full Speed clock.

bool CLOCK_EnableUsbfs0HostClock(*clock_usbfs_src_t* src, uint32_t freq)

Enable USB HOST FS clock.

Parameters

- src – : clock source
- freq – clock frequency Enable USB HOST Full Speed clock.

void CLOCK_EnableOstimer32kClock(void)

Enable the OSTIMER 32k clock.

Returns

Nothing

void CLOCK_XtalHfCapabankTrim(int32_t pi32_hfXtalIecLoadpF_x100, int32_t pi32_hfXtalPPcbParCappF_x100, int32_t pi32_hfXtalNPcbParCappF_x100)

Sets board-specific trim values for High Frequency crystal oscillator.

Note: Following default Values can be used: pi32_32MfXtalIecLoadpF_x100 Load capacitance, pF x 100 : 600 pi32_32MfXtalPPcbParCappF_x100 PCB +ve parasitic capacitance, pF x 100 : 20 pi32_32MfXtalNPcbParCappF_x100 PCB -ve parasitic capacitance, pF x 100 : 40

Parameters

- pi32_hfXtalIecLoadpF_x100 – Load capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
- pi32_hfXtalPPcbParCappF_x100 – PCB +ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
- pi32_hfXtalNPcbParCappF_x100 – PCB -ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120

Returns

none

void CLOCK_Xtal32khzCapabankTrim(int32_t pi32_32kfXtalIecLoadpF_x100, int32_t pi32_32kfXtalPPcbParCappF_x100, int32_t pi32_32kfXtalNPcbParCappF_x100)

Sets board-specific trim values for 32kHz XTAL.

Note: Following default Values can be used: pi32_32kfXtalIecLoadpF_x100 Load capacitance, pF x 100 : 600 pi32_32kfXtalPPcbParCappF_x100 PCB +ve parasitic capacitance, pF x 100 : 40 pi32_32kfXtalNPcbParCappF_x100 PCB -ve parasitic capacitance, pF x 100 : 40

Parameters

- pi32_32kfXtalIecLoadpF_x100 – Load capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
- pi32_32kfXtalPPcbParCappF_x100 – PCB +ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120
- pi32_32kfXtalNPcbParCappF_x100 – PCB -ve parasitic capacitance, pF x 100. For example, 6pF becomes 600, 1.2pF becomes 120

Returns

none

void CLOCK_FroHfTrim(void)

Initialize the trim value for FRO HF.

Returns

none

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.3.3.

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.

CLOCK_USR_CFG_PLL_CONFIG_CACHE_COUNT

User-defined the size of cache for CLOCK_PllGetConfig() function.

Once define this MACRO to be non-zero value, CLOCK_PllGetConfig() function would cache the recent calculation and accelerate the execution to get the right settings.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

ROM_CLOCKS

Clock ip name array for ROM.

SRAM_CLOCKS

Clock ip name array for SRAM.

FLASH_CLOCKS

Clock ip name array for FLASH.

FMC_CLOCKS

Clock ip name array for FMC.

INPUTMUX_CLOCKS

Clock ip name array for INPUTMUX.

IOCON_CLOCKS

Clock ip name array for IOCON.

GPIO_CLOCKS

Clock ip name array for GPIO.

PINT_CLOCKS

Clock ip name array for PINT.

GINT_CLOCKS

Clock ip name array for GINT.

DMA_CLOCKS

Clock ip name array for DMA.

CRC_CLOCKS

Clock ip name array for CRC.

WWDT_CLOCKS

Clock ip name array for WWDT.

RTC_CLOCKS

Clock ip name array for RTC.

MAILBOX_CLOCKS

Clock ip name array for Mailbox.

LPADC_CLOCKS

Clock ip name array for LPADC.

LPDAC_CLOCKS

Clock ip name array for DAC.

MRT_CLOCKS

Clock ip name array for MRT.

OSTIMER_CLOCKS

Clock ip name array for OSTIMER.

SCT_CLOCKS

Clock ip name array for SCT0.

MCAN_CLOCKS

Clock ip name array for MCAN.

UTICK_CLOCKS

Clock ip name array for UTICK.

FLEXCOMM_CLOCKS

Clock ip name array for FLEXCOMM.

LPUART_CLOCKS

Clock ip name array for LPUART.

BI2C_CLOCKS

Clock ip name array for BI2C.

LPSPI_CLOCKS

Clock ip name array for LSPI.

FLEXI2S_CLOCKS

Clock ip name array for FLEXI2S.

CTIMER_CLOCKS

Clock ip name array for CTIMER.

COMP_CLOCKS

Clock ip name array for COMP.

FREQME_CLOCKS

Clock ip name array for FREQME.

CDOG_CLOCKS

Clock ip name array for CDOG.

RNG_CLOCKS

Clock ip name array for RNG.

USBHMR0_CLOCKS

Clock ip name array for USBHMR0.

USBHSL0_CLOCKS

Clock ip name array for USBHSL0.

ANALOGCTRL_CLOCKS

Clock ip name array for ANALOGCTRL.

HS_LSPI_CLOCKS

Clock ip name array for HS_LSPI.

GPIO_SEC_CLOCKS

Clock ip name array for GPIO_SEC.

GPIO_SEC_INT_CLOCKS

Clock ip name array for GPIO_SEC_INT.

USB_D_CLOCKS

Clock ip name array for USB.

SYSCTL_CLOCKS

Clock ip name array for SYSCTL.

DMIC_CLOCKS

Clock ip name array for DMIC.

PWM_CLOCKS

Clock ip name array for PWM.

ENC_CLOCKS

Clock ip name array for ENC.

OPAMP_CLOCKS

Clock ip name array for OPAMP.

VREF_CLOCKS

Clock ip name array for VREF.

FLEXSPI_CLOCKS

Clock ip name array for FLEXSPI.

CACHE64_CLOCKS

Clock ip name array for Cache64.

I3C_CLOCKS

Clock ip name array for I3C.

HSCMP_CLOCKS

 Clock ip name array for HSCMP.

POWERQUAD_CLOCKS

 Clock ip name array for PowerQuad.

AOI_CLOCKS

 Clock ip name array for AOI.

CLK_GATE_REG_OFFSET_SHIFT

 Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

CLK_GATE_REG_OFFSET_MASK

CLK_GATE_BIT_SHIFT_SHIFT

CLK_GATE_BIT_SHIFT_MASK

CLK_GATE_DEFINE(reg_offset, bit_shift)

CLK_GATE_ABSTRACT_REG_OFFSET(x)

CLK_GATE_ABSTRACT_BITS_SHIFT(x)

AHB_CLK_CTRL0

AHB_CLK_CTRL1

AHB_CLK_CTRL2

AHB_CLK_CTRL3

REG_PWM0SUBCTL

REG_PWM1SUBCTL

BUS_CLK

 Peripherals clock source definition.

I2C0_CLK_SRC

CLK_ATTACH_ID(mux, sel, pos)

 Clock Mux Switches The encoding is as follows each connection identified is 32bits wide while 24bits are valuable starting from LSB upwards.

 [4 bits for choice, 0 means invalid choice] [8 bits mux ID]*

MUX_A(mux, sel)

MUX_B(mux, sel, selector)

GET_ID_ITEM(connection)

GET_ID_NEXT_ITEM(connection)

GET_ID_ITEM_MUX(connection)

GET_ID_ITEM_SEL(connection)

GET_ID_SELECTOR(connection)

CM_SYSTICKCLKSEL0

CM_TRACECLKSEL

CM_CTIMERCLKSEL0
CM_CTIMERCLKSEL1
CM_CTIMERCLKSEL2
CM_CTIMERCLKSEL3
CM_CTIMERCLKSEL4
CM_MAINCLKSELA
CM_MAINCLKSELB
CM_CLKOUTCLKSEL
CM_PLL0CLKSEL
CM_PLL1CLKSEL
CM_MCANCLKSEL
CM_ADC0CLKSEL
CM_USB0CLKSEL
CM_FXCOMCLKSEL0
CM_FXCOMCLKSEL1
CM_FXCOMCLKSEL2
CM_FXCOMCLKSEL3
CM_FXCOMCLKSEL4
CM_FXCOMCLKSEL5
CM_FXCOMCLKSEL6
CM_FXCOMCLKSEL7
CM_HSLSPICLKSEL
CM_MCLKCLKSEL
CM_SCTCLKSEL
CM_ADC1CLKSEL
CM_DAC0CLKSEL
CM_DAC1CLKSEL
CM_DAC2CLKSEL
CM_FLEXSPICLKSEL
CM_PLLCLKDIVSEL
CM_I3CFCLKSEL
CM_I3CFCLKSTCSEL
CM_I3CFCLKSSEL

CM_DMICCLKSEL

CM_FC32KCLKSEL

CM_FRGCLKSEL0

CM_FRGCLKSEL1

CM_FRGCLKSEL2

CM_FRGCLKSEL3

CM_FRGCLKSEL4

CM_FRGCLKSEL5

CM_FRGCLKSEL6

CM_FRGCLKSEL7

CM_OSTIMERCLKSEL

CM_RTCOSC32KCLKSEL

PLL_CONFIGFLAG_USEINRATE

PLL configuration structure flags for 'flags' field These flags control how the PLL configuration function sets up the PLL setup structure.

When the PLL_CONFIGFLAG_USEINRATE flag is selected, the 'InputRate' field in the configuration structure must be assigned with the expected PLL frequency. If the PLL_CONFIGFLAG_USEINRATE is not used, 'InputRate' is ignored in the configuration function and the driver will determine the PLL rate from the currently selected PLL source. This flag might be used to configure the PLL input clock more accurately when using the WDT oscillator or a more dynamic CLKIN source.

When the PLL_CONFIGFLAG_FORCENOFRACT flag is selected, the PLL hardware for the automatic bandwidth selection, Spread Spectrum (SS) support, and fractional M-divider are not used.

Flag to use InputRate in PLL configuration structure for setup

PLL_CONFIGFLAG_FORCENOFRACT

Force non-fractional output mode, PLL output will not use the fractional, automatic bandwidth, or SS hardware

PLL_SETUPFLAG_POWERUP

PLL setup structure flags for 'flags' field These flags control how the PLL setup function sets up the PLL.

Setup will power on the PLL after setup

PLL_SETUPFLAG_WAITLOCK

Setup will wait for PLL lock, implies the PLL will be powered on

PLL_SETUPFLAG_ADGVOLT

Optimize system voltage for the new PLL rate

PLL_SETUPFLAG_USEFEEDBACKDIV2

Use feedback divider by 2 in divider path

uint32_t desiredRate

Desired PLL rate in Hz

uint32_t inputRate

PLL input clock in Hz, only used if PLL_CONFIGFLAG_USEINRATE flag is set

uint32_t flags

PLL configuration flags, Or'ed value of PLL_CONFIGFLAG_* definitions

ss_progmodfm_t ss_mf

SS Programmable modulation frequency, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag

ss_progmoddp_t ss_mr

SS Programmable frequency modulation depth, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag

ss_modwvctrl_t ss_mc

SS Modulation waveform control, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag

bool mfDither

false for fixed modulation frequency or true for dithering, only applicable when not using PLL_CONFIGFLAG_FORCENOFRACT flag

uint32_t pllctrl

PLL control register PLL0CTRL

uint32_t pllndec

PLL NDEC register PLL0NDEC

uint32_t pllpdec

PLL PDEC register PLL0PDEC

uint32_t pllmdec

PLL MDEC registers PLL0PDEC

uint32_t pllssc[2]

PLL SSCTL registers PLL0SSCG

uint32_t pllRate

Actual PLL rate

uint32_t flags

PLL setup flags, Or'ed value of PLL_SETUPFLAG_* definitions

struct __pll_config

#include <fsl_clock.h> PLL configuration structure.

This structure can be used to configure the settings for a PLL setup structure. Fill in the desired configuration for the PLL and call the PLL setup function to fill in a PLL setup structure.

struct __pll_setup

#include <fsl_clock.h> PLL0 setup structure This structure can be used to pre-build a PLL setup configuration at run-time and quickly set the PLL to the configuration. It can be populated with the PLL setup function. If powering up or waiting for PLL lock, the PLL input clock source should be configured prior to PLL setup.

2.10 CMP: Analog Comparator Driver

void CMP_Init(const *cmp_config_t* *config)

CMP initialization.

This function enables the CMP module and do necessary settings.

Parameters

- config – Pointer to the configuration structure.

void CMP_Deinit(void)

CMP deinitialization.

This function gates the clock for CMP module.

void CMP_GetDefaultConfig(*cmp_config_t* *config)

Initializes the CMP user configuration structure.

This function initializes the user configuration structure to these default values.

```
config->enableHysteresis    = true;
config->enableLowPower      = true;
config->filterClockDivider  = kCMP_FilterClockDivide1;
config->filterSampleMode    = kCMP_FilterSampleMode0;
```

Parameters

- config – Pointer to the configuration structure.

static inline void CMP_SetInputChannels(uint8_t positiveChannel, uint8_t negativeChannel)

void CMP_SetVREF(const *cmp_vref_config_t* *config)

Configures the VREFINPUT.

Parameters

- config – Pointer to the configuration structure.

static inline bool CMP_GetOutput(void)

Get CMP compare output.

Returns

The output result. true: voltage on positive side is greater than negative side.

false: voltage on positive side is lower than negative side.

static inline void CMP_EnableInterrupt(uint32_t type)

CMP enable interrupt.

Parameters

- type – CMP interrupt type. See “_cmp_interrupt_type”.

static inline void CMP_DisableInterrupt(void)

CMP disable interrupt.

static inline void CMP_ClearInterrupt(void)

CMP clear interrupt.

static inline void CMP_EnableFilteredInterruptSource(bool enable)

Select which Analog comparator output (filtered or un-filtered) is used for interrupt detection.

Note: : When CMP is configured as the wakeup source in power down mode, this function must use the raw output as the interrupt source, that is, call this function and set parameter enable to false.

Parameters

- `enable` – `false`: Select Analog Comparator raw output (unfiltered) as input for interrupt detection. `true`: Select Analog Comparator filtered output as input for interrupt detection.

static inline bool CMP_GetPreviousInterruptStatus(void)

Get CMP interrupt status before interrupt enable.

Returns

Interrupt status. `true`: interrupt pending, `false`: no interrupt pending.

static inline bool CMP_GetInterruptStatus(void)

Get CMP interrupt status after interrupt enable.

Returns

Interrupt status. `true`: interrupt pending, `false`: no interrupt pending.

static inline void CMP_FilterSampleConfig(*cmp_filtercfg_samplemode_t* filterSampleMode,
cmp_filtercfg_clkdiv_t filterClockDivider)

CMP Filter Sample Config.

This function allows the users to configure the sampling mode and clock divider of the CMP Filter.

Parameters

- `filterSampleMode` – CMP Select filter sample mode
- `filterClockDivider` – CMP Set filter clock divider

FSL_CMP_DRIVER_VERSION

Driver version 2.2.1.

enum _cmp_input_mux

CMP input mux for positive and negative sides.

Values:

enumerator kCMP_InputVREF

Cmp input from VREF.

enumerator kCMP_Input1

Cmp input source 1.

enumerator kCMP_Input2

Cmp input source 2.

enumerator kCMP_Input3

Cmp input source 3.

enumerator kCMP_Input4

Cmp input source 4.

enumerator kCMP_Input5

Cmp input source 5.

enum _cmp_interrupt_type

CMP interrupt type.

Values:

enumerator kCMP_EdgeDisable

Disable edge interrupt.

enumerator kCMP_EdgeRising
Interrupt on falling edge.

enumerator kCMP_EdgeFalling
Interrupt on rising edge.

enumerator kCMP_EdgeRisingFalling
Interrupt on both rising and falling edges.

enumerator kCMP_LevelDisable
Disable level interrupt.

enumerator kCMP_LevelHigh
Interrupt on high level.

enumerator kCMP_LevelLow
Interrupt on low level.

enum _cmp_vref_source
CMP Voltage Reference source.

Values:

enumerator KCMP_VREFSourceVDDA
Select VDDA as VREF.

enumerator KCMP_VREFSourceInternalVREF
Select internal VREF as VREF.

enum _cmp_filtercgf_samplemode
CMP Filter sample mode.

Values:

enumerator kCMP_FilterSampleMode0
Bypass mode. Filtering is disabled.

enumerator kCMP_FilterSampleMode1
Filter 1 clock period.

enumerator kCMP_FilterSampleMode2
Filter 2 clock period.

enumerator kCMP_FilterSampleMode3
Filter 3 clock period.

enum _cmp_filtercgf_clkdiv
CMP Filter clock divider.

Values:

enumerator kCMP_FilterClockDivide1
Filter clock period duration equals 1 analog comparator clock period.

enumerator kCMP_FilterClockDivide2
Filter clock period duration equals 2 analog comparator clock period.

enumerator kCMP_FilterClockDivide4
Filter clock period duration equals 4 analog comparator clock period.

enumerator kCMP_FilterClockDivide8
Filter clock period duration equals 8 analog comparator clock period.

enumerator kCMP_FilterClockDivide16

Filter clock period duration equals 16 analog comparator clock period.

enumerator kCMP_FilterClockDivide32

Filter clock period duration equals 32 analog comparator clock period.

enumerator kCMP_FilterClockDivide64

Filter clock period duration equals 64 analog comparator clock period.

typedef enum *_cmp_vref_source* cmp_vref_source_t

CMP Voltage Reference source.

typedef struct *_cmp_vref_config* cmp_vref_config_t

typedef enum *_cmp_filtercgf_samplemode* cmp_filtercgf_samplemode_t

CMP Filter sample mode.

typedef enum *_cmp_filtercgf_clkdiv* cmp_filtercgf_clkdiv_t

CMP Filter clock divider.

typedef struct *_cmp_config* cmp_config_t

CMP configuration structure.

struct *_cmp_vref_config*

#include <fsl_cmp.h>

Public Members

cmp_vref_source_t vrefSource

Reference voltage source.

uint8_t vrefValue

Reference voltage step. Available range is 0-31. Per step equals to VREFINPUT/31.

struct *_cmp_config*

#include <fsl_cmp.h> CMP configuration structure.

Public Members

bool enableHysteresis

Enable hysteresis.

bool enableLowPower

Enable low power mode.

2.11 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- Version 2.0.4
 - Release peripheral from reset if necessary in init function.

- Version 2.0.3
 - Fix MISRA issues
- Version 2.0.2
 - Fix MISRA issues
- Version 2.0.1
 - move DATA and DATALL macro definition from header file to source file

enum `_crc_bits`

CRC bit width.

Values:

enumerator `kCrcBits16`

Generate 16-bit CRC code

enumerator `kCrcBits32`

Generate 32-bit CRC code

enum `_crc_result`

CRC result type.

Values:

enumerator `kCrcFinalChecksum`

CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

enumerator `kCrcIntermediateChecksum`

CRC data register read value is intermediate checksum (raw value). Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for `CRC_Init()` to continue adding data to this checksum.

typedef enum `_crc_bits` `crc_bits_t`

CRC bit width.

typedef enum `_crc_result` `crc_result_t`

CRC result type.

typedef struct `_crc_config` `crc_config_t`

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void `CRC_Init(CRC_Type *base, const crc_config_t *config)`

Enables and configures the CRC peripheral module.

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

- `base` – CRC peripheral address.
- `config` – CRC module configuration structure.

static inline void `CRC_Deinit(CRC_Type *base)`

Disables the CRC peripheral module.

This function disables the clock gate in the SIM module for the CRC peripheral.

Parameters

- `base` – CRC peripheral address.

`void CRC_GetDefaultConfig(crc_config_t *config)`

Loads default values to the CRC protocol configuration structure.

Loads default values to the CRC protocol configuration structure. The default values are as follows.

```
config->polynomial = 0x1021;
config->seed = 0xFFFF;
config->reflectIn = false;
config->reflectOut = false;
config->complementChecksum = false;
config->crcBits = kCrcBits16;
config->crcResult = kCrcFinalChecksum;
```

Parameters

- config – CRC protocol configuration structure.

`void CRC_WriteData(CRC_Type *base, const uint8_t *data, size_t dataSize)`

Writes data to the CRC module.

Writes input data buffer bytes to the CRC data register. The configured type of transpose is applied.

Parameters

- base – CRC peripheral address.
- data – Input data stream, MSByte in data[0].
- dataSize – Size in bytes of the input data buffer.

`uint32_t CRC_Get32bitResult(CRC_Type *base)`

Reads the 32-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 32-bit checksum, after configured transpose and complement operations.

`uint16_t CRC_Get16bitResult(CRC_Type *base)`

Reads a 16-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 16-bit checksum, after configured transpose and complement operations.

`CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT`

Default configuration structure filled by `CRC_GetDefaultConfig()`. Use CRC16-CCIT-FALSE as default.

`struct _crc_config`

`#include <fsl_crc.h>` CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

CRC Polynomial, MSBit first. Example polynomial: $0x1021 = 1\_0000\_0010\_0001 = x^{12} + x^5 + 1$

uint32_t seed

Starting checksum value

bool reflectIn

Reflect bits on input.

bool reflectOut

Reflect bits on output.

bool complementChecksum

True if the result shall be complement of the actual checksum.

crc_bits_t crcBits

Selects 16- or 32- bit CRC protocol.

crc_result_t crcResult

Selects final or intermediate checksum return from CRC_Get16bitResult() or CRC_Get32bitResult()

2.12 CTIMER: Standard counter/timers

void CTIMER_Init(CTIMER_Type *base, const *ctimer_config_t* *config)

Un-gates the clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application before using the driver.

Parameters

- base – Ctimer peripheral base address
- config – Pointer to the user configuration structure.

void CTIMER_Deinit(CTIMER_Type *base)

Gates the timer clock.

Parameters

- base – Ctimer peripheral base address

void CTIMER_GetDefaultConfig(*ctimer_config_t* *config)

Fills in the timers configuration structure with the default settings.

The default values are:

```
config->mode = kCTIMER_TimerMode;
config->input = kCTIMER_Capture_0;
config->prescale = 0;
```

Parameters

- config – Pointer to the user configuration structure.

```
status_t CTIMER_SetupPwmPeriod(CTIMER_Type *base, const ctimer_match_t
                                pwmPeriodChannel, ctimer_match_t matchChannel,
                                uint32_t pwmPeriod, uint32_t pulsePeriod, bool enableInt)
```

Configures the PWM signal parameters.

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note: When setting PWM output from multiple output pins, all should use the same PWM period

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- pwmPeriod – PWM period match value
- pulsePeriod – Pulse width match value
- enableInt – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

Returns

kStatus_Success on success kStatus_Fail If matchChannel is equal to pwmPeriodChannel; this channel is reserved to set the PWM cycle If PWM pulse width register value is larger than 0xFFFFFFFF.

```
status_t CTIMER_SetupPwm(CTIMER_Type *base, const ctimer_match_t pwmPeriodChannel,
                          ctimer_match_t matchChannel, uint8_t dutyCyclePercent, uint32_t
                          pwmFreq_Hz, uint32_t srcClock_Hz, bool enableInt)
```

Configures the PWM signal parameters.

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note: When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER_SetupPwmPeriod to set up the PWM with high resolution.

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- dutyCyclePercent – PWM pulse width; the value should be between 0 to 100
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – Timer counter clock in Hz
- enableInt – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

```
static inline void CTIMER_UpdatePwmPulsePeriod(CTIMER_Type *base, ctimer_match_t
                                              matchChannel, uint32_t pulsePeriod)
```

Updates the pulse period of an active PWM signal.

Parameters

- base – Ctimer peripheral base address
- matchChannel – Match pin to be used to output the PWM signal
- pulsePeriod – New PWM pulse width match value

```
status_t CTIMER_UpdatePwmDutycycle(CTIMER_Type *base, const ctimer_match_t
                                   pwmPeriodChannel, ctimer_match_t matchChannel,
                                   uint8_t dutyCyclePercent)
```

Updates the duty cycle of an active PWM signal.

Note: Please use CTIMER_SetupPwmPeriod to update the PWM with high resolution. This function can manually assign the specified channel to set the PWM cycle.

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- dutyCyclePercent – New PWM pulse width; the value should be between 0 to 100

Returns

kStatus_Success on success kStatus_Fail If PWM pulse width register value is larger than 0xFFFFFFFF.

```
static inline void CTIMER_EnableInterrupts(CTIMER_Type *base, uint32_t mask)
```

Enables the selected Timer interrupts.

Parameters

- base – Ctimer peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ctimer_interrupt_enable_t*

```
static inline void CTIMER_DisableInterrupts(CTIMER_Type *base, uint32_t mask)
```

Disables the selected Timer interrupts.

Parameters

- base – Ctimer peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ctimer_interrupt_enable_t*

```
static inline uint32_t CTIMER_GetEnabledInterrupts(CTIMER_Type *base)
```

Gets the enabled Timer interrupts.

Parameters

- base – Ctimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *ctimer_interrupt_enable_t*


```
static inline uint32_t CTIMER_GetStatusFlags(CTIMER_Type *base)
```

Gets the Timer status flags.

Parameters

- base – Ctimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `ctimer_status_flags_t`

```
static inline void CTIMER_ClearStatusFlags(CTIMER_Type *base, uint32_t mask)
```

Clears the Timer status flags.

Parameters

- base – Ctimer peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `ctimer_status_flags_t`

```
static inline void CTIMER_StartTimer(CTIMER_Type *base)
```

Starts the Timer counter.

Parameters

- base – Ctimer peripheral base address

```
static inline void CTIMER_StopTimer(CTIMER_Type *base)
```

Stops the Timer counter.

Parameters

- base – Ctimer peripheral base address

FSL_CTIMER_DRIVER_VERSION

Version 2.3.3

```
enum _ctimer_capture_channel
```

List of Timer capture channels.

Values:

enumerator `kCTIMER_Capture_0`

Timer capture channel 0

enumerator `kCTIMER_Capture_1`

Timer capture channel 1

enumerator `kCTIMER_Capture_3`

Timer capture channel 3

```
enum _ctimer_capture_edge
```

List of capture edge options.

Values:

enumerator `kCTIMER_Capture_RiseEdge`

Capture on rising edge

enumerator `kCTIMER_Capture_FallEdge`

Capture on falling edge

enumerator `kCTIMER_Capture_BothEdge`

Capture on rising and falling edge

enum `_ctimer_match`

List of Timer match registers.

Values:

enumerator `kCTIMER_Match_0`

Timer match register 0

enumerator `kCTIMER_Match_1`

Timer match register 1

enumerator `kCTIMER_Match_2`

Timer match register 2

enumerator `kCTIMER_Match_3`

Timer match register 3

enum `_ctimer_external_match`

List of external match.

Values:

enumerator `kCTIMER_External_Match_0`

External match 0

enumerator `kCTIMER_External_Match_1`

External match 1

enumerator `kCTIMER_External_Match_2`

External match 2

enumerator `kCTIMER_External_Match_3`

External match 3

enum `_ctimer_match_output_control`

List of output control options.

Values:

enumerator `kCTIMER_Output_NoAction`

No action is taken

enumerator `kCTIMER_Output_Clear`

Clear the EM bit/output to 0

enumerator `kCTIMER_Output_Set`

Set the EM bit/output to 1

enumerator `kCTIMER_Output_Toggle`

Toggle the EM bit/output

enum `_ctimer_timer_mode`

List of Timer modes.

Values:

enumerator `kCTIMER_TimerMode`

enumerator `kCTIMER_IncreaseOnRiseEdge`

enumerator `kCTIMER_IncreaseOnFallEdge`

enumerator `kCTIMER_IncreaseOnBothEdge`

enum `_ctimer_interrupt_enable`

List of Timer interrupts.

Values:

enumerator `kCTIMER_Match0InterruptEnable`

Match 0 interrupt

enumerator `kCTIMER_Match1InterruptEnable`

Match 1 interrupt

enumerator `kCTIMER_Match2InterruptEnable`

Match 2 interrupt

enumerator `kCTIMER_Match3InterruptEnable`

Match 3 interrupt

enum `_ctimer_status_flags`

List of Timer flags.

Values:

enumerator `kCTIMER_Match0Flag`

Match 0 interrupt flag

enumerator `kCTIMER_Match1Flag`

Match 1 interrupt flag

enumerator `kCTIMER_Match2Flag`

Match 2 interrupt flag

enumerator `kCTIMER_Match3Flag`

Match 3 interrupt flag

enum `ctimer_callback_type_t`

Callback type when registering for a callback. When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Values:

enumerator `kCTIMER_SingleCallback`

Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently

enumerator `kCTIMER_MultipleCallback`

Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

typedef enum `_ctimer_capture_channel` `ctimer_capture_channel_t`

List of Timer capture channels.

typedef enum `_ctimer_capture_edge` `ctimer_capture_edge_t`

List of capture edge options.

typedef enum `_ctimer_match` `ctimer_match_t`

List of Timer match registers.

typedef enum `_ctimer_external_match` `ctimer_external_match_t`

List of external match.

typedef enum `_ctimer_match_output_control` `ctimer_match_output_control_t`

List of output control options.

```
typedef enum _ctimer_timer_mode ctimer_timer_mode_t
```

List of Timer modes.

```
typedef enum _ctimer_interrupt_enable ctimer_interrupt_enable_t
```

List of Timer interrupts.

```
typedef enum _ctimer_status_flags ctimer_status_flags_t
```

List of Timer flags.

```
typedef void (*ctimer_callback_t)(uint32_t flags)
```

```
typedef struct _ctimer_match_config ctimer_match_config_t
```

Match configuration.

This structure holds the configuration settings for each match register.

```
typedef struct _ctimer_config ctimer_config_t
```

Timer configuration structure.

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the CTIMER_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

```
void CTIMER_SetupMatch(CTIMER_Type *base, ctimer_match_t matchChannel, const  
                      ctimer_match_config_t *config)
```

Setup the match register.

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

- base – Ctimer peripheral base address
- matchChannel – Match register to configure
- config – Pointer to the match configuration structure

```
uint32_t CTIMER_GetOutputMatchStatus(CTIMER_Type *base, uint32_t matchChannel)
```

Get the status of output match.

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

Parameters

- base – Ctimer peripheral base address
- matchChannel – External match channel, user can obtain the status of multiple match channels at the same time by using the logic of “|” enumeration *ctimer_external_match_t*

Returns

The mask of external match channel status flags. Users need to use the *_ctimer_external_match* type to decode the return variables.

```
void CTIMER_SetupCapture(CTIMER_Type *base, ctimer_capture_channel_t capture,  
                        ctimer_capture_edge_t edge, bool enableInt)
```

Setup the capture.

Parameters

- base – Ctimer peripheral base address
- capture – Capture channel to configure

- edge – Edge on the channel that will trigger a capture
- enableInt – Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

static inline uint32_t CTIMER_GetTimerCountValue(CTIMER_Type *base)

Get the timer count value from TC register.

Parameters

- base – Ctimer peripheral base address.

Returns

return the timer count value.

void CTIMER_RegisterCallBack(CTIMER_Type *base, *ctimer_callback_t* *cb_func,
ctimer_callback_type_t cb_type)

Register callback.

This function configures CTimer Callback in following modes:

- Single Callback: cb_func should be pointer to callback function pointer
For example: *ctimer_callback_t* *ctimer_callback* = *pwm_match_callback*;
CTIMER_RegisterCallBack(CTIMER, &*ctimer_callback*, kCTIMER_SingleCallback);
- Multiple Callback: cb_func should be pointer to array of callback function pointers
Each element corresponds to Interrupt Flag in IR register. For example: *ctimer_callback_t* *ctimer_callback_table*[] = {
ctimer_match0_callback, NULL, NULL, *ctimer_match3_callback*, NULL, NULL, NULL, NULL};
CTIMER_RegisterCallBack(CTIMER, &*ctimer_callback_table*[0], kCTIMER_MultipleCallback);

Parameters

- base – Ctimer peripheral base address
- cb_func – Pointer to callback function pointer
- cb_type – callback function type, singular or multiple

static inline void CTIMER_Reset(CTIMER_Type *base)

Reset the counter.

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

- base – Ctimer peripheral base address

static inline void CTIMER_SetPrescale(CTIMER_Type *base, uint32_t prescale)

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

Parameters

- base – Ctimer peripheral base address
- prescale – Prescale value

static inline uint32_t CTIMER_GetCaptureValue(CTIMER_Type *base, *ctimer_capture_channel_t*
capture)

Get capture channel value.

Get the counter/timer value on the corresponding capture channel.

Parameters

- base – Ctimer peripheral base address

- capture – Select capture channel

Returns

The timer count capture value.

```
static inline void CTIMER__EnableResetMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                    match, bool enable)
```

Enable reset match channel.

Set the specified match channel reset operation.

Parameters

- base – Ctimer peripheral base address
- match – match channel used
- enable – Enable match channel reset operation.

```
static inline void CTIMER__EnableStopMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                    match, bool enable)
```

Enable stop match channel.

Set the specified match channel stop operation.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable match channel stop operation.

```
static inline void CTIMER__EnableMatchChannelReload(CTIMER_Type *base, ctimer_match_t
                                                    match, bool enable)
```

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable .

```
static inline void CTIMER__EnableRisingEdgeCapture(CTIMER_Type *base,
                                                    ctimer_capture_channel_t capture, bool
                                                    enable)
```

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable rising edge capture.

```
static inline void CTIMER__EnableFallingEdgeCapture(CTIMER_Type *base,
                                                    ctimer_capture_channel_t capture, bool
                                                    enable)
```

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable falling edge capture.

```
static inline void CTIMER_SetShadowValue(CTIMER_Type *base, ctimer_match_t match,
                                         uint32_t matchvalue)
```

Set the specified match shadow channel.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- matchvalue – Reload the value of the corresponding match register.

```
struct __ctimer_match_config
```

#include <fsl_ctimer.h> Match configuration.

This structure holds the configuration settings for each match register.

Public Members

uint32_t matchValue

This is stored in the match register

bool enableCounterReset

true: Match will reset the counter false: Match will not reset the counter

bool enableCounterStop

true: Match will stop the counter false: Match will not stop the counter

ctimer_match_output_control_t outControl

Action to be taken on a match on the EM bit/output

bool outPinInitState

Initial value of the EM bit/output

bool enableInterrupt

true: Generate interrupt upon match false: Do not generate interrupt on match

```
struct __ctimer_config
```

#include <fsl_ctimer.h> Timer configuration structure.

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the CTIMER_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

ctimer_timer_mode_t mode

Timer mode

ctimer_capture_channel_t input

Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC

uint32_t prescale

Prescale value

2.13 DAC: Digital-to-Analog Converter Driver

`void DAC_Init(LPDAC_Type *base, const dac_config_t *config)`

Initialize the DAC module with common configuration.

The clock will be enabled in this function.

Parameters

- `base` – DAC peripheral base address.
- `config` – Pointer to configuration structure.

`void DAC_GetDefaultConfig(dac_config_t *config)`

Get the default settings for initialization's configuration.

This function initializes the user configuration structure to a default value. The default values are:

```
config->fifoWatermarkLevel = 0U;  
config->fifoTriggerMode = kDAC_FIFOTriggerByHardwareMode;  
config->fifoWorkMode = kDAC_FIFODisabled;  
config->enableLowPowerMode = false;  
config->referenceVoltageSource = kDAC_ReferenceVoltageSourceAlt1;
```

Parameters

- `config` – Pointer to configuration structure.

`void DAC_Deinit(LPDAC_Type *base)`

De-initialize the DAC module.

The clock will be disabled in this function.

Parameters

- `base` – DAC peripheral base address.

`static inline void DAC_SetReset(LPDAC_Type *base, uint32_t mask)`

Assert the reset control to part hardware.

This function is to assert the reset control to part hardware. Responding part hardware would remain reset until cleared by software.

Parameters

- `base` – DAC peripheral base address.
- `mask` – The reset control mask, see to `_dac_reset_control_t`.

`static inline void DAC_ClearReset(LPDAC_Type *base, uint32_t mask)`

Clear the reset control to part hardware.

This function is to clear the reset control to part hardware. Responding part hardware would work after the reset control is cleared by software.

Parameters

- `base` – DAC peripheral base address.
- `mask` – The reset control mask, see to `_dac_reset_control_t`.

`static inline void DAC_Enable(LPDAC_Type *base, bool enable)`

Enable the DAC hardware system or not.

This function is to start the Programmable Reference Generator operation or not.

Parameters

- base – DAC peripheral base address.
- enable – Assertion of indicated event.

static inline void DAC_EnableInterrupts(LPDAC_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – DAC peripheral base address.
- mask – Mask value of indicated interrupt events. See to `_dac_interrupt_enable`.

static inline void DAC_DisableInterrupts(LPDAC_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – DAC peripheral base address.
- mask – Mask value of indicated interrupt events. See to `_dac_interrupt_enable`.

static inline void DAC_EnableDMA(LPDAC_Type *base, uint32_t mask, bool enable)

Enable the DMA switchers or not.

Parameters

- base – DAC peripheral base address.
- mask – Mask value of indicated DMA request. See to `_dac_dma_enable`.
- enable – Enable the DMA or not.

static inline uint32_t DAC_GetStatusFlags(LPDAC_Type *base)

Get status flags of DAC module.

Parameters

- base – DAC peripheral base address.

Returns

Mask value of status flags. See to `_dac_status_flags`.

static inline void DAC_ClearStatusFlags(LPDAC_Type *base, uint32_t flags)

Clear status flags of DAC module.

Parameters

- base – DAC peripheral base address.
- flags – Mask value of status flags to be cleared. See to `_dac_status_flags`.

static inline void DAC_SetData(LPDAC_Type *base, uint32_t value)

Set data into the entry of FIFO buffer.

Parameters

- base – DAC peripheral base address.
- value – Setting value into FIFO buffer.

static inline uint32_t DAC_GetFIFOWritePointer(LPDAC_Type *base)

Get the value of the FIFO write pointer.

Parameters

- base – DAC peripheral base address.

Returns

Current value of the FIFO write pointer.

static inline uint32_t DAC_GetFIFOReadPointer(LPDAC_Type *base)

Get the value of the FIFO read pointer.

Parameters

- base – DAC peripheral base address.

Returns

Current value of the FIFO read pointer.

static inline void DAC_DoSoftwareTriggerFIFO(LPDAC_Type *base)

Do software trigger to FIFO when in software mode.

Parameters

- base – DAC peripheral base address.

FSL_DAC_DRIVER_VERSION

DAC driver version 2.1.3.

DAC reset control.

Values:

enumerator kDAC_ResetFIFO

Resets the FIFO pointers and flags.

enumerator kDAC_ResetLogic

Resets all DAC registers and internal logic.

DAC interrupts.

Values:

enumerator kDAC_FIFOFullInterruptEnable

FIFO full interrupt enable.

enumerator kDAC_FIFOWatermarkInterruptEnable

FIFO watermark interrupt enable.

enumerator kDAC_FIFOSwingBackInterruptEnable

Swing back one cycle complete interrupt enable.

enumerator kDAC_FIFOOverflowInterruptEnable

FIFO overflow interrupt enable.

enumerator kDAC_FIFOUnderflowInterruptEnable

FIFO underflow interrupt enable.

enumerator kDAC_PeriodTriggerCompleteInterruptEnable

Period trigger mode conversion complete interrupt enable

DAC DMA switchers.

Values:

enumerator kDAC_FIFODMAEnable

FIFO empty DMA enable.

enumerator `kDAC_FIFOWatermarkDMAEnable`
FIFO watermark DMA enable.

DAC status flags.

Values:

enumerator `kDAC_FIFOUnderflowFlag`

This flag means that there is a new trigger after the buffer is empty. The FIFO read pointer will not increase in this case and the data sent to DAC analog conversion will not changed. This flag is cleared by writing a 1 to it.

enumerator `kDAC_FIFOverflowFlag`

This flag indicates that data is intended to write into FIFO after the buffer is full. The writer pointer will not increase in this case. The extra data will not be written into the FIFO. This flag is cleared by writing a 1 to it.

enumerator `kDAC_FIFOSwingBackFlag`

This flag indicates that the DAC has completed one period of conversion in swing back mode. It means that the read pointer has increased to the top (write pointer) once and then decreased to zero once. For example, after three data is written to FIFO, the writer pointer is now 3. Then, if continually triggered, the read pointer will swing like: 0-1-2-1-0-1-2-, and so on. After the fourth trigger, the flag is set. This flag is cleared by writing a 1 to it.

enumerator `kDAC_FIFOWatermarkFlag`

This field is set if the remaining data in FIFO is less than or equal to the setting value of watermark. By writing data into FIFO by DMA or CPU, this flag is cleared automatically when the data in FIFO is more than the setting value of watermark.

enumerator `kDAC_FIFOEmptyFlag`

FIFO empty flag.

enumerator `kDAC_FIFOFullFlag`

FIFO full flag.

enumerator `kDAC_PeriodTriggerCompleteFlag`

Period trigger mode conversion complete flag.

enum `_dac_fifo_trigger_mode`

DAC FIFO trigger mode.

Values:

enumerator `kDAC_FIFOTriggerByHardwareMode`

Buffer would be triggered by hardware.

enumerator `kDAC_FIFOTriggerBySoftwareMode`

Buffer would be triggered by software.

enum `_dac_fifo_work_mode`

DAC FIFO work mode.

Values:

enumerator `kDAC_FIFODisabled`

FIFO mode is disabled and buffer mode is enabled. Any data written to DATA[DATA] goes to buffer then goes to conversion.

enumerator `kDAC_FIFOWorkAsNormalMode`

FIFO mode is enabled. Data will be first read from FIFO to buffer then goes to conversion.

enumerator kDAC_FIFOWorkAsSwingMode

In swing mode, the read pointer swings between the writer pointer and zero. That is, the trigger increases the read pointer till reach the writer pointer and decreases the read pointer till zero, and so on. The FIFO empty/full/watermark flag will not update during swing back mode.

enumerator kDAC_FIFOWorkAsPeriodTriggerMode

In periodic trigger mode, user only needs to send the first trigger. Then after every [PTG_PERIOD+1] RCLK cycles, DAC will be automatically triggered by internal trigger. There will be [PTG_NUM] internal triggers, thus in total [PTG_NUM+1] conversions including the first trigger sent by user. User can terminate the current conversion queue by clearing the GCR[PTGEN] bit. Then, after the current conversion is completed, the conversion is terminated and the PTGCOCO flag is set. If PCR[PTG_NUM] is set to zero, there will be infinite triggers following the first hardware/software trigger, until the GCR[PTGEN] is cleared by software. In any case, the conversion can be terminated by FIFORST/SWRST.

enumerator kDAC_FIFOWorkAsPeriodTriggerAndSwingMode

Periodically trigger DAC and swing back.

enum _dac_reference_voltage_source

DAC reference voltage source.

Values:

enumerator kDAC_ReferenceVoltageSourceAlt1

The DAC selects VREFH_INT as the reference voltage.

enumerator kDAC_ReferenceVoltageSourceAlt2

The DAC selects VREFH_EXT as the reference voltage.

enum _dac_reference_current_source

Values:

enumerator kDAC_ReferenceCurrentSourcePtat

enumerator kDAC_ReferenceCurrentSourceZtc

typedef enum _dac_fifo_trigger_mode dac_fifo_trigger_mode_t

DAC FIFO trigger mode.

typedef enum _dac_fifo_work_mode dac_fifo_work_mode_t

DAC FIFO work mode.

typedef enum _dac_reference_voltage_source dac_reference_voltage_source_t

DAC reference voltage source.

typedef enum _dac_reference_current_source dac_reference_current_source_t

typedef struct _dac_config dac_config_t

DAC configuration structure.

struct _dac_config

#include <fsl_dac.h> DAC configuration structure.

Public Members

uint32_t fifoWatermarkLevel

FIFO's watermark, the max value can be the hardware FIFO size.

dac_fifo_trigger_mode_t fifoTriggerMode

Select the trigger mode for FIFO.

dac_fifo_work_mode_t fifoWorkMode

Select the work mode for FIFO.

bool enableOpampBuffer

Opamp is used as buffer.

bool enableLowerLowPowerMode

Enable the lower low power mode.

uint32_t periodicTriggerNumber

There will be 'periodicTriggerNumber' internal triggers following the first hardware/software trigger. So there will be 'periodicTriggerNumber + 1' conversions in total. If set to zero, there will be infinite triggers following the first hw/sw trigger, until the GCR[PTGEN] is cleared.

uint32_t periodicTriggerWidth

Control the periodic trigger frequency. There will be 'periodicTriggerWidth + 1' RCLK cycles between each periodic trigger. The periodic trigger frequency should be configured to not larger than the analog conversion speed.

uint32_t syncTime

RCLK cycles before data latch. accessible range is 0-15. It is used to configure the DAC sync cycles which is helpful to reduce glitch on the output. The sync time is (LATCH_CYC+1) RCLK cycles. User should configure this register according to the RCLK frequency. The recommended sync time is at least 40ns.

dac_reference_current_source_t referenceCurrentSource

Select the internal reference current source.

dac_reference_voltage_source_t referenceVoltageSource

Select the reference voltage source.

2.14 DMA: Direct Memory Access Controller Driver

void DMA_Init(DMA_Type *base)

Initializes DMA peripheral.

This function enable the DMA clock, set descriptor table and enable DMA peripheral.

Parameters

- base – DMA peripheral base address.

void DMA_Deinit(DMA_Type *base)

Deinitializes DMA peripheral.

This function gates the DMA clock.

Parameters

- base – DMA peripheral base address.

void DMA_InstallDescriptorMemory(DMA_Type *base, void *addr)

Install DMA descriptor memory.

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, although current DMA driver has a default DMA descriptor buffer, but it support one DMA descriptor for one channel only.

Parameters

- base – DMA base address.
- addr – DMA descriptor address

static inline bool DMA_ChannelIsActive(DMA_Type *base, uint32_t channel)

Return whether DMA channel is processing transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

True for active state, false otherwise.

static inline bool DMA_ChannelIsBusy(DMA_Type *base, uint32_t channel)

Return whether DMA channel is busy.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

True for busy state, false otherwise.

static inline void DMA_EnableChannelInterrupts(DMA_Type *base, uint32_t channel)

Enables the interrupt source for the DMA transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_DisableChannelInterrupts(DMA_Type *base, uint32_t channel)

Disables the interrupt source for the DMA transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_EnableChannel(DMA_Type *base, uint32_t channel)

Enable DMA channel.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_DisableChannel(DMA_Type *base, uint32_t channel)

Disable DMA channel.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_EnableChannelPeriphRq(DMA_Type *base, uint32_t channel)

Set PERIPHREQEN of channel configuration register.

Parameters

- base – DMA peripheral base address.

- channel – DMA channel number.

static inline void DMA_DisableChannelPeriphRq(DMA_Type *base, uint32_t channel)

Get PERIPHREQEN value of channel configuration register.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

True for enabled PeriphRq, false for disabled.

void DMA_ConfigureChannelTrigger(DMA_Type *base, uint32_t channel, dma_channel_trigger_t *trigger)

Set trigger settings of DMA channel.

Deprecated:

Do not use this function. It has been superceded by DMA_SetChannelConfig.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.
- trigger – trigger configuration.

void DMA_SetChannelConfig(DMA_Type *base, uint32_t channel, dma_channel_trigger_t *trigger, bool isPeriph)

set channel config.

This function provide a interface to configure channel configuration registers.

Parameters

- base – DMA base address.
- channel – DMA channel number.
- trigger – channel configurations structure.
- isPeriph – true is periph request, false is not.

static inline uint32_t DMA_SetChannelXferConfig(bool reload, bool clrTrig, bool intA, bool intB, uint8_t width, uint8_t srcInc, uint8_t dstInc, uint32_t bytes)

DMA channel xfer transfer configurations.

Parameters

- reload – true is reload link descriptor after current exhaust, false is not
- clrTrig – true is clear trigger status, wait software trigger, false is not
- intA – enable interruptA
- intB – enable interruptB
- width – transfer width
- srcInc – source address interleave size
- dstInc – destination address interleave size
- bytes – transfer bytes

Returns

The vaule of xfer config

uint32_t DMA_GetRemainingBytes(DMA_Type *base, uint32_t channel)

Gets the remaining bytes of the current DMA descriptor transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

The number of bytes which have not been transferred yet.

static inline void DMA_SetChannelPriority(DMA_Type *base, uint32_t channel, dma_priority_t priority)

Set priority of channel configuration register.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.
- priority – Channel priority value.

static inline dma_priority_t DMA_GetChannelPriority(DMA_Type *base, uint32_t channel)

Get priority of channel configuration register.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

Channel priority value.

static inline void DMA_SetChannelConfigValid(DMA_Type *base, uint32_t channel)

Set channel configuration valid.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_DoChannelSoftwareTrigger(DMA_Type *base, uint32_t channel)

Do software trigger for the channel.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_LoadChannelTransferConfig(DMA_Type *base, uint32_t channel, uint32_t xfer)

Load channel transfer configurations.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.
- xfer – transfer configurations.


```
void DMA_CreateDescriptor(dma_descriptor_t *desc, dma_xfercfg_t *xfercfg, void *srcAddr, void *dstAddr, void *nextDesc)
```

Create application specific DMA descriptor to be used in a chain in transfer.

Deprecated:

Do not use this function. It has been superceded by DMA_SetupDescriptor.

Parameters

- desc – DMA descriptor address.
- xfercfg – Transfer configuration for DMA descriptor.
- srcAddr – Address of last item to transmit
- dstAddr – Address of last item to receive.
- nextDesc – Address of next descriptor in chain.

```
void DMA_SetupDescriptor(dma_descriptor_t *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)
```

setup dma descriptor

Note: This function do not support configure wrap descriptor.

Parameters

- desc – DMA descriptor address.
- xfercfg – Transfer configuration for DMA descriptor.
- srcStartAddr – Start address of source address.
- dstStartAddr – Start address of destination address.
- nextDesc – Address of next descriptor in chain.

```
void DMA_SetupChannelDescriptor(dma_descriptor_t *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc, dma_burst_wrap_t wrapType, uint32_t burstSize)
```

setup dma channel descriptor

Note: This function support configure wrap descriptor.

Parameters

- desc – DMA descriptor address.
- xfercfg – Transfer configuration for DMA descriptor.
- srcStartAddr – Start address of source address.
- dstStartAddr – Start address of destination address.
- nextDesc – Address of next descriptor in chain.
- wrapType – burst wrap type.
- burstSize – burst size, reference `_dma_burst_size`.

```
void DMA_LoadChannelDescriptor(DMA_Type *base, uint32_t channel, dma_descriptor_t *descriptor)
```

load channel transfer decriptor.

This function can be used to load desscriptor to driver internal channel descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

- a. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
DMA_Init(DMA0);
DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);
DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);
DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (dma_descriptor_t *)desc);
DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);
while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))
{}
```

Parameters

- base – DMA base address.
- channel – DMA channel.
- descriptor – configured DMA descriptor.

void DMA_AbortTransfer(*dma_handle_t* *handle)

Abort running transfer by handle.

This function aborts DMA transfer specified by handle.

Parameters

- handle – DMA handle pointer.

void DMA_CreateHandle(*dma_handle_t* *handle, DMA_Type *base, uint32_t channel)

Creates the DMA handle.

This function is called if using transaction API for DMA. This function initializes the internal state of DMA handle.

Parameters

- handle – DMA handle pointer. The DMA handle stores callback function and parameters.
- base – DMA peripheral base address.
- channel – DMA channel number.

void DMA_SetCallback(*dma_handle_t* *handle, *dma_callback* callback, void *userData)

Installs a callback function for the DMA transfer.

This callback is called in DMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

- handle – DMA handle pointer.
- callback – DMA callback function pointer.
- userData – Parameter for callback function.

void DMA_PrepareTransfer(*dma_transfer_config_t* *config, void *srcAddr, void *dstAddr, uint32_t byteWidth, uint32_t transferBytes, *dma_transfer_type_t* type, void *nextDesc)

Prepares the DMA transfer structure.

Deprecated:

Do not use this function. It has been superseded by DMA_PrepareChannelTransfer. 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, so the source address must be 4 bytes aligned, or it shall result in source address error(SAE).

Parameters

- config – The user configuration structure of type `dma_transfer_t`.
- srcAddr – DMA transfer source address.
- dstAddr – DMA transfer destination address.
- byteWidth – DMA transfer destination address width(bytes).
- transferBytes – DMA transfer bytes to be transferred.
- type – DMA transfer type.
- nextDesc – Chain custom descriptor to transfer.

```
void DMA_PrepareChannelTransfer(dma_channel_config_t *config, void *srcStartAddr, void  
                               *dstStartAddr, uint32_t xferCfg, dma_transfer_type_t type,  
                               dma_channel_trigger_t *trigger, void *nextDesc)
```

Prepare channel transfer configurations.

This function used to prepare channel transfer configurations.

Parameters

- config – Pointer to DMA channel transfer configuration structure.
- srcStartAddr – source start address.
- dstStartAddr – destination start address.
- xferCfg – xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
- type – transfer type.
- trigger – DMA channel trigger configurations.
- nextDesc – address of next descriptor.

```
status_t DMA_SubmitTransfer(dma_handle_t *handle, dma_transfer_config_t *config)
```

Submits the DMA transfer request.

Deprecated:

Do not use this function. It has been superceded by `DMA_SubmitChannelTransfer`.

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

- handle – DMA handle pointer.
- config – Pointer to DMA transfer configuration structure.

Return values

- `kStatus_DMA_Success` – It means submit transfer request succeed.
- `kStatus_DMA_QueueFull` – It means TCD queue is full. Submit transfer request is not allowed.

- `kStatus_DMA_Busy` – It means the given channel is busy, need to submit request later.

`void DMA_SubmitChannelTransferParameter(dma_handle_t *handle, uint32_t xferCfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)`

Submit channel transfer paramter directly.

This function used to configure channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

- for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload, clrTrig,
↪ intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, NULL);
DMA_StartTransfer(handle)
```

- for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload, clrTrig,
↪ intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
DMA_StartTransfer(handle);
```

Parameters

- `handle` – Pointer to DMA handle.
- `xferCfg` – xfer configuration, user can reference `DMA_CHANNEL_XFER` about to how to get `xferCfg` value.
- `srcStartAddr` – source start address.
- `dstStartAddr` – destination start address.
- `nextDesc` – address of next descriptor.

`void DMA_SubmitChannelDescriptor(dma_handle_t *handle, dma_descriptor_t *descriptor)`

Submit channel descriptor.

This function used to configure channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this function is typical for the ping pong case:

- a. for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```
define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelDescriptor(handle, nextDesc0);
DMA_StartTransfer(handle);
```

Parameters

- handle – Pointer to DMA handle.
- descriptor – descriptor to submit.

status_t DMA_SubmitChannelTransfer(*dma_handle_t* *handle, *dma_channel_config_t* *config)

Submits the DMA channel transfer request.

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

- a. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,NULL);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
```

- b. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,
↪ nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
```

- c. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,
↪nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
```

Parameters

- handle – DMA handle pointer.
- config – Pointer to DMA transfer configuration structure.

Return values

- kStatus_DMA_Success – It means submit transfer request succeed.
- kStatus_DMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_DMA_Busy – It means the given channel is busy, need to submit request later.

void DMA_StartTransfer(*dma_handle_t* *handle)

DMA start transfer.

This function enables the channel request. User can call this function after submitting the transfer request. It will trigger transfer start with software trigger only when hardware trigger is not used.

Parameters

- handle – DMA handle pointer.

void DMA_IRQHandle(DMA_Type *base)

DMA IRQ handler for descriptor transfer complete.

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

Parameters

- base – DMA base address.

FSL_DMA_DRIVER_VERSION

DMA driver version.

Version 2.5.3.

_dma_transfer_status DMA transfer status

Values:

enumerator kStatus_DMA_Busy

Channel is busy and can't handle the transfer request.

`_dma_addr_interleave_size` dma address interleave size

Values:

enumerator `kDMA__AddressInterleave0xWidth`
dma source/destination address no interleave

enumerator `kDMA__AddressInterleave1xWidth`
dma source/destination address interleave 1xwidth

enumerator `kDMA__AddressInterleave2xWidth`
dma source/destination address interleave 2xwidth

enumerator `kDMA__AddressInterleave4xWidth`
dma source/destination address interleave 3xwidth

`_dma_transfer_width` dma transfer width

Values:

enumerator `kDMA__Transfer8BitWidth`
dma channel transfer bit width is 8 bit

enumerator `kDMA__Transfer16BitWidth`
dma channel transfer bit width is 16 bit

enumerator `kDMA__Transfer32BitWidth`
dma channel transfer bit width is 32 bit

enum `_dma_priority`

DMA channel priority.

Values:

enumerator `kDMA__ChannelPriority0`
Highest channel priority - priority 0

enumerator `kDMA__ChannelPriority1`
Channel priority 1

enumerator `kDMA__ChannelPriority2`
Channel priority 2

enumerator `kDMA__ChannelPriority3`
Channel priority 3

enumerator `kDMA__ChannelPriority4`
Channel priority 4

enumerator `kDMA__ChannelPriority5`
Channel priority 5

enumerator `kDMA__ChannelPriority6`
Channel priority 6

enumerator `kDMA__ChannelPriority7`
Lowest channel priority - priority 7

enum `_dma_int`

DMA interrupt flags.

Values:

enumerator kDMA_IntA
DMA interrupt flag A

enumerator kDMA_IntB
DMA interrupt flag B

enumerator kDMA_IntError
DMA interrupt flag error

enum _dma_trigger_type
DMA trigger type.

Values:

enumerator kDMA_NoTrigger
Trigger is disabled

enumerator kDMA_LowLevelTrigger
Low level active trigger

enumerator kDMA_HighLevelTrigger
High level active trigger

enumerator kDMA_FallingEdgeTrigger
Falling edge active trigger

enumerator kDMA_RisingEdgeTrigger
Rising edge active trigger

_dma_burst_size DMA burst size

Values:

enumerator kDMA_BurstSize1
burst size 1 transfer

enumerator kDMA_BurstSize2
burst size 2 transfer

enumerator kDMA_BurstSize4
burst size 4 transfer

enumerator kDMA_BurstSize8
burst size 8 transfer

enumerator kDMA_BurstSize16
burst size 16 transfer

enumerator kDMA_BurstSize32
burst size 32 transfer

enumerator kDMA_BurstSize64
burst size 64 transfer

enumerator kDMA_BurstSize128
burst size 128 transfer

enumerator kDMA_BurstSize256
burst size 256 transfer

enumerator kDMA_BurstSize512
burst size 512 transfer

enumerator kDMA__BurstSize1024

burst size 1024 transfer

enum __dma__trigger__burst

DMA trigger burst.

Values:

enumerator kDMA__SingleTransfer

Single transfer

enumerator kDMA__LevelBurstTransfer

Burst transfer driven by level trigger

enumerator kDMA__EdgeBurstTransfer1

Perform 1 transfer by edge trigger

enumerator kDMA__EdgeBurstTransfer2

Perform 2 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer4

Perform 4 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer8

Perform 8 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer16

Perform 16 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer32

Perform 32 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer64

Perform 64 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer128

Perform 128 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer256

Perform 256 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer512

Perform 512 transfers by edge trigger

enumerator kDMA__EdgeBurstTransfer1024

Perform 1024 transfers by edge trigger

enum __dma__burst__wrap

DMA burst wrapping.

Values:

enumerator kDMA__NoWrap

Wrapping is disabled

enumerator kDMA__SrcWrap

Wrapping is enabled for source

enumerator kDMA__DstWrap

Wrapping is enabled for destination

enumerator kDMA__SrcAndDstWrap

Wrapping is enabled for source and destination

enum *_dma_transfer_type*

DMA transfer type.

Values:

enumerator *kDMA_MemoryToMemory*

Transfer from memory to memory (increment source and destination)

enumerator *kDMA_PeripheralToMemory*

Transfer from peripheral to memory (increment only destination)

enumerator *kDMA_MemoryToPeripheral*

Transfer from memory to peripheral (increment only source)

enumerator *kDMA_StaticToStatic*

Peripheral to static memory (do not increment source or destination)

typedef struct *_dma_descriptor* *dma_descriptor_t*

DMA descriptor structure.

typedef struct *_dma_xfercfg* *dma_xfercfg_t*

DMA transfer configuration.

typedef enum *_dma_priority* *dma_priority_t*

DMA channel priority.

typedef enum *_dma_int* *dma_irq_t*

DMA interrupt flags.

typedef enum *_dma_trigger_type* *dma_trigger_type_t*

DMA trigger type.

typedef enum *_dma_trigger_burst* *dma_trigger_burst_t*

DMA trigger burst.

typedef enum *_dma_burst_wrap* *dma_burst_wrap_t*

DMA burst wrapping.

typedef enum *_dma_transfer_type* *dma_transfer_type_t*

DMA transfer type.

typedef struct *_dma_channel_trigger* *dma_channel_trigger_t*

DMA channel trigger.

typedef struct *_dma_channel_config* *dma_channel_config_t*

DMA channel trigger.

typedef struct *_dma_transfer_config* *dma_transfer_config_t*

DMA transfer configuration.

typedef void (**dma_callback*)(struct *_dma_handle* **handle*, void **userData*, bool *transferDone*,
uint32_t *intmode*)

Define Callback function for DMA.

typedef struct *_dma_handle* *dma_handle_t*

DMA transfer handle structure.

DMA_MAX_TRANSFER_COUNT

DMA max transfer size.

FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE

DMA channel numbers.

DMA head link descriptor table align size

DMA_ALLOCATE_HEAD_DESCRIPTOR(name, number)

DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate descriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_HEAD_DESCRIPTOR_AT_NONCACHEABLE(name, number)

DMA head descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate descriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_LINK_DESCRIPTOR(name, number)

DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate descriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_LINK_DESCRIPTOR_AT_NONCACHEABLE(name, number)

DMA link descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate descriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_DATA_TRANSFER_BUFFER(name, width)

DMA transfer buffer address need to align with the transfer width.

DMA_CHANNEL_GROUP(channel)

DMA_CHANNEL_INDEX(base, channel)

DMA_COMMON_REG_GET(base, channel, reg)

DMA linked descriptor address align size.

DMA_COMMON_CONST_REG_GET(base, channel, reg)

DMA_COMMON_REG_SET(base, channel, reg, value)

DMA_DESCRIPTOR_END_ADDRESS(start, inc, bytes, width)

DMA descriptor end address calculate.

Parameters

- start – start address
- inc – address interleave size
- bytes – transfer bytes

- width – transfer width

DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes)

struct __dma_descriptor

#include <fsl_dma.h> DMA descriptor structure.

Public Members

volatile uint32_t xfercfg

Transfer configuration

void *srcEndAddr

Last source address of DMA transfer

void *dstEndAddr

Last destination address of DMA transfer

void *linkToNextDesc

Address of next DMA descriptor in chain

struct __dma_xfercfg

#include <fsl_dma.h> DMA transfer configuration.

Public Members

bool valid

Descriptor is ready to transfer

bool reload

Reload channel configuration register after current descriptor is exhausted

bool swtrig

Perform software trigger. Transfer if fired when 'valid' is set

bool clrtrig

Clear trigger

bool intA

Raises IRQ when transfer is done and set IRQA status register flag

bool intB

Raises IRQ when transfer is done and set IRQB status register flag

uint8_t byteWidth

Byte width of data to transfer

uint8_t srcInc

Increment source address by 'srcInc' x 'byteWidth'

uint8_t dstInc

Increment destination address by 'dstInc' x 'byteWidth'

uint16_t transferCount

Number of transfers

struct __dma_channel_trigger

#include <fsl_dma.h> DMA channel trigger.

Public Members

dma_trigger_type_t type

Select hardware trigger as edge triggered or level triggered.

dma_trigger_burst_t burst

Select whether hardware triggers cause a single or burst transfer.

dma_burst_wrap_t wrap

Select wrap type, source wrap or dest wrap, or both.

struct *_dma_channel_config*

#include <fsl_dma.h> DMA channel trigger.

Public Members

void *srcStartAddr

Source data address

void *dstStartAddr

Destination data address

void *nextDesc

Chain custom descriptor

uint32_t xferCfg

channel transfer configurations

dma_channel_trigger_t *trigger

DMA trigger type

bool isPeriph

select the request type

struct *_dma_transfer_config*

#include <fsl_dma.h> DMA transfer configuration.

Public Members

uint8_t *srcAddr

Source data address

uint8_t *dstAddr

Destination data address

uint8_t *nextDesc

Chain custom descriptor

dma_xfercfg_t xfercfg

Transfer options

bool isPeriph

DMA transfer is driven by peripheral

struct *_dma_handle*

#include <fsl_dma.h> DMA transfer handle structure.

Public Members

dma_callback callback

Callback function. Invoked when transfer of descriptor with interrupt flag finishes

void *userData

Callback function parameter

DMA_Type *base

DMA peripheral base address

uint8_t channel

DMA channel number

2.15 DMIC: Digital Microphone

2.16 DMIC DMA Driver

status_t DMIC_TransferCreateHandleDMA(DMIC_Type *base, *dmic_dma_handle_t* *handle, *dmic_dma_transfer_callback_t* callback, void *userData, *dma_handle_t* *rxDmaHandle)

Initializes the DMIC handle which is used in transactional functions.

Parameters

- base – DMIC peripheral base address.
- handle – Pointer to *dmic_dma_handle_t* structure.
- callback – Callback function.
- userData – User data.
- rxDmaHandle – User-requested DMA handle for RX DMA transfer.

status_t DMIC_TransferReceiveDMA(DMIC_Type *base, *dmic_dma_handle_t* *handle, *dmic_transfer_t* *xfer, uint32_t channel)

Receives data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – USART peripheral base address.
- handle – Pointer to *usart_dma_handle_t* structure.
- xfer – DMIC DMA transfer structure. See *dmic_transfer_t*.
- channel – DMIC start channel number.

Return values

kStatus_Success –

void DMIC_TransferAbortReceiveDMA(DMIC_Type *base, *dmic_dma_handle_t* *handle)

Aborts the received data using DMA.

This function aborts the received data using DMA.

Parameters

- base – DMIC peripheral base address
- handle – Pointer to *dmic_dma_handle_t* structure

```
status_t DMIC_TransferGetReceiveCountDMA(DMIC_Type *base, dmic_dma_handle_t *handle,
                                         uint32_t *count)
```

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – DMIC peripheral base address.
- handle – DMIC 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 DMIC_InstallDMADescriptorMemory(dmic_dma_handle_t *handle, void *linkAddr, size_t
                                     linkNum)
```

Install DMA descriptor memory.

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, it should be called after DMIC_TransferCreateHandleDMA. User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

- handle – Pointer to DMA channel transfer handle.
- linkAddr – DMA link descriptor address.
- linkNum – DMA link descriptor number.

```
FSL_DMIC_DMA_DRIVER_VERSION
```

DMIC DMA driver version 2.4.2.

```
typedef struct _dmic_transfer dmic_transfer_t
```

DMIC transfer structure.

```
typedef struct _dmic_dma_handle dmic_dma_handle_t
```

```
typedef void (*dmic_dma_transfer_callback_t)(DMIC_Type *base, dmic_dma_handle_t *handle,
status_t status, void *userData)
```

DMIC transfer callback function.

```
struct _dmic_transfer
```

#include <fsl_dmic_dma.h> DMIC transfer structure.

Public Members

```
void *data
```

The buffer of data to be transfer.

```
uint8_t dataWidth
```

DMIC support 16bit/32bit

```
size_t dataSize
```

The byte count to be transfer.

uint8_t dataAddrInterleaveSize
destination address interleave size

struct *_dmic_transfer* *linkTransfer
use to support link transfer

struct *_dmic_dma_handle*
#include <fsl_dmic_dma.h> DMIC DMA handle.

Public Members

DMIC_Type *base
DMIC peripheral base address.

dma_handle_t *rxDmaHandle
The DMA RX channel used.

dmic_dma_transfer_callback_t callback
Callback function.

void *userData
DMIC callback function parameter.

size_t transferSize
Size of the data to receive.

volatile uint8_t state
Internal state of DMIC DMA transfer

uint32_t channel
DMIC channel used.

bool isChannelValid
DMIC channel initialization flag

dma_descriptor_t *desLink
descriptor pool pointer

size_t linkNum
number of descriptor in descriptors pool

2.17 DMIC Driver

uint32_t DMIC_GetInstance(DMIC_Type *base)
Get the DMIC instance from peripheral base address.

Parameters

- base – DMIC peripheral base address.

Returns

DMIC instance.

void DMIC_Init(DMIC_Type *base)
Turns DMIC Clock on.

Parameters

- base – : DMIC base

Returns

Nothing

void DMIC_DeInit(DMIC_Type *base)

Turns DMIC Clock off.

Parameters

- base – : DMIC base

Returns

Nothing

void DMIC_SetOperationMode(DMIC_Type *base, operation_mode_t mode)

Set DMIC operating mode.

Deprecated:

Do not use this function. It has been superceded by DMIC_EnableChannelInterrupt, DMIC_EnableChannelDma.

Parameters

- base – : The base address of DMIC interface
- mode – : DMIC mode

Returns

Nothing

void DMIC_Use2fs(DMIC_Type *base, bool use2fs)

Configure Clock scaling.

Parameters

- base – : The base address of DMIC interface
- use2fs – : clock scaling

Returns

Nothing

void DMIC_CfgChannelDc(DMIC_Type *base, dmic_channel_t channel, dc_removal_t dc_cut_level, uint32_t post_dc_gain_reduce, bool saturate16bit)

Configure DMIC channel.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- dc_cut_level – : dc_removal_t, Cut off Frequency
- post_dc_gain_reduce – : Fine gain adjustment in the form of a number of bits to downshift.
- saturate16bit – : If selects 16-bit saturation.

static inline void DMIC_EnableChannelSignExtend(DMIC_Type *base, dmic_channel_t channel, bool enable)

Enbale channel sign extend which allows processing of 24bit audio data on 32bit machines.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

- enable – : true is enable sign extend, false is disable sign extend

```
void DMIC_ConfigChannel(DMIC_Type *base, dmic_channel_t channel, stereo_side_t side,  
                       dmic_channel_config_t *channel_config)
```

Configure DMIC channel.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- side – : *stereo_side_t*, choice of left or right
- channel_config – : Channel configuration

Returns

Nothing

```
void DMIC_EnableChannel(DMIC_Type *base, uint32_t channelmask)
```

Enable a particular channel.

Parameters

- base – : The base address of DMIC interface
- channelmask – reference *_dmic_channel_mask*

Returns

Nothing

```
void DMIC_FifoChannel(DMIC_Type *base, uint32_t channel, uint32_t trig_level, uint32_t  
                     enable, uint32_t resetn)
```

Configure fifo settings for DMIC channel.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- trig_level – : FIFO trigger level
- enable – : FIFO level
- resetn – : FIFO reset

Returns

Nothing

```
static inline void DMIC_EnableChannelInterrupt(DMIC_Type *base, dmic_channel_t channel,  
                                                bool enable)
```

Enable a particular channel interrupt request.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection
- enable – : true is enable, false is disable

```
static inline void DMIC_EnableChannelDma(DMIC_Type *base, dmic_channel_t channel, bool  
                                         enable)
```

Enable a particular channel dma request.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection

- enable – : true is enable, false is disable

```
static inline void DMIC__EnableChannelFifo(DMIC_Type *base, dmic_channel_t channel, bool enable)
```

Enable a particular channel fifo.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection
- enable – : true is enable, false is disable

```
static inline void DMIC__DoFifoReset(DMIC_Type *base, dmic_channel_t channel)
```

Channel fifo reset.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection

```
static inline uint32_t DMIC__FifoGetStatus(DMIC_Type *base, uint32_t channel)
```

Get FIFO status.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

Returns

FIFO status

```
static inline void DMIC__FifoClearStatus(DMIC_Type *base, uint32_t channel, uint32_t mask)
```

Clear FIFO status.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- mask – : Bits to be cleared

Returns

FIFO status

```
static inline uint32_t DMIC__FifoGetData(DMIC_Type *base, uint32_t channel)
```

Get FIFO data.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

Returns

FIFO data

```
static inline uint32_t DMIC__FifoGetAddress(DMIC_Type *base, uint32_t channel)
```

Get FIFO address.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

Returns

FIFO data

`void DMIC__ResetChannelDecimator(DMIC_Type *base, uint32_t channelMask, bool reset)`
DMIC channel Decimator reset.

Parameters

- `base` – : The base address of DMIC interface
- `channelMask` – : DMIC channel mask, reference `_dmic_channel_mask`
- `reset` – : `true` is reset decimator, `false` is release decimator.

`static inline void DMIC__EnableChannelGlobalSync(DMIC_Type *base, uint32_t channelMask, uint32_t syncCounter)`

Enable DMIC channel global sync function.

Parameters

- `base` – : The base address of DMIC interface
- `channelMask` – : DMIC channel mask, reference `_dmic_channel_mask`
- `syncCounter` – : sync counter will trigger a pulse whenever count reaches `CCOUNTVAL`. If `CCOUNTVAL` is set to 0, there will be a pulse on every cycle

`static inline void DMIC__DisableChannelGlobalSync(DMIC_Type *base, uint32_t channelMask)`
Disble DMIC channel global sync function.

Parameters

- `base` – : The base address of DMIC interface
- `channelMask` – : DMIC channel mask, reference `_dmic_channel_mask`

`void DMIC__EnableIntCallback(DMIC_Type *base, dmic_callback_t cb)`
Enable callback.

This function enables the interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

- `base` – Base address of the DMIC peripheral.
- `cb` – callback Pointer to store callback function.

Return values

None. –

`void DMIC__DisableIntCallback(DMIC_Type *base, dmic_callback_t cb)`
Disable callback.

This function disables the interrupt for the selected DMIC peripheral.

Parameters

- `base` – Base address of the DMIC peripheral.
- `cb` – callback Pointer to store callback function..

Return values

None. –

`static inline void DMIC__SetGainNoiseEstHwvad(DMIC_Type *base, uint32_t value)`
Sets the gain value for the noise estimator.

Parameters

- `base` – DMIC base pointer
- `value` – gain value for the noise estimator.

Return values

None. –

static inline void DMIC__SetGainSignalEstHwvad(DMIC_Type *base, uint32_t value)

Sets the gain value for the signal estimator.

Parameters

- base – DMIC base pointer
- value – gain value for the signal estimator.

Return values

None. –

static inline void DMIC__SetFilterCtrlHwvad(DMIC_Type *base, uint32_t value)

Sets the hwvad filter cutoff frequency parameter.

Parameters

- base – DMIC base pointer
- value – cut off frequency value.

Return values

None. –

static inline void DMIC__SetInputGainHwvad(DMIC_Type *base, uint32_t value)

Sets the input gain of hwvad.

Parameters

- base – DMIC base pointer
- value – input gain value for hwvad.

Return values

None. –

static inline void DMIC__CtrlClrIntrHwvad(DMIC_Type *base, bool st10)

Clears hwvad internal interrupt flag.

Parameters

- base – DMIC base pointer
- st10 – bit value.

Return values

None. –

static inline void DMIC__FilterResetHwvad(DMIC_Type *base, bool rstt)

Resets hwvad filters.

Parameters

- base – DMIC base pointer
- rstt – Reset bit value.

Return values

None. –

static inline uint16_t DMIC__GetNoiseEnvlpEst(DMIC_Type *base)

Gets the value from output of the filter z7.

Parameters

- base – DMIC base pointer

Return values

output – of filter z7.

void DMIC_HwvadEnableIntCallback(DMIC_Type *base, *dmic_hwvad_callback_t* vadcb)

Enable hwvad callback.

This function enables the hwvad interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

- base – Base address of the DMIC peripheral.
- vadcb – callback Pointer to store callback function.

Return values

None. –

void DMIC_HwvadDisableIntCallback(DMIC_Type *base, *dmic_hwvad_callback_t* vadcb)

Disable callback.

This function disables the hwvad interrupt for the selected DMIC peripheral.

Parameters

- base – Base address of the DMIC peripheral.
- vadcb – callback Pointer to store callback function..

Return values

None. –

FSL_DMIC_DRIVER_VERSION

DMIC driver version 2.3.3.

_dmic_status DMIC transfer status.

Values:

enumerator kStatus_DMIC_Busy

DMIC is busy

enumerator kStatus_DMIC_Idle

DMIC is idle

enumerator kStatus_DMIC_OverRunError

DMIC over run Error

enumerator kStatus_DMIC_UnderRunError

DMIC under run Error

enum __operation__mode

DMIC different operation modes.

Values:

enumerator kDMIC_OperationModeInterrupt

Interrupt mode

enumerator kDMIC_OperationModeDma

DMA mode

enum __stereo__side

DMIC left/right values.

Values:

enumerator kDMIC_Left

Left Stereo channel

enumerator kDMIC_Right

Right Stereo channel

enum pdm_div_t

DMIC Clock pre-divider values.

Values:

enumerator kDMIC_PdmDiv1

DMIC pre-divider set in divide by 1

enumerator kDMIC_PdmDiv2

DMIC pre-divider set in divide by 2

enumerator kDMIC_PdmDiv3

DMIC pre-divider set in divide by 3

enumerator kDMIC_PdmDiv4

DMIC pre-divider set in divide by 4

enumerator kDMIC_PdmDiv6

DMIC pre-divider set in divide by 6

enumerator kDMIC_PdmDiv8

DMIC pre-divider set in divide by 8

enumerator kDMIC_PdmDiv12

DMIC pre-divider set in divide by 12

enumerator kDMIC_PdmDiv16

DMIC pre-divider set in divide by 16

enumerator kDMIC_PdmDiv24

DMIC pre-divider set in divide by 24

enumerator kDMIC_PdmDiv32

DMIC pre-divider set in divide by 32

enumerator kDMIC_PdmDiv48

DMIC pre-divider set in divide by 48

enumerator kDMIC_PdmDiv64

DMIC pre-divider set in divide by 64

enumerator kDMIC_PdmDiv96

DMIC pre-divider set in divide by 96

enumerator kDMIC_PdmDiv128

DMIC pre-divider set in divide by 128

enum __compensation

Pre-emphasis Filter coefficient value for 2FS and 4FS modes.

Values:

enumerator kDMIC_CompValueZero

Compensation 0

enumerator kDMIC_CompValueNegativePoint16

Compensation -0.16

enumerator kDMIC__CompValueNegativePoint15
Compensation -0.15

enumerator kDMIC__CompValueNegativePoint13
Compensation -0.13

enum __dc_removal

DMIC DC filter control values.

Values:

enumerator kDMIC__DcNoRemove
Flat response no filter

enumerator kDMIC__DcCut155
Cut off Frequency is 155 Hz

enumerator kDMIC__DcCut78
Cut off Frequency is 78 Hz

enumerator kDMIC__DcCut39
Cut off Frequency is 39 Hz

enum __dmic_channel

DMIC Channel number.

Values:

enumerator kDMIC__Channel0
DMIC channel 0

enumerator kDMIC__Channel1
DMIC channel 1

enumerator kDMIC__ChannelMAX
Maximum number of DMIC channels

__dmic_channel_mask DMIC Channel mask.

Values:

enumerator kDMIC__EnableChannel0
DMIC channel 0 mask

enumerator kDMIC__EnableChannel1
DMIC channel 1 mask

enum __dmic_phy_sample_rate

DMIC and decimator sample rates.

Values:

enumerator kDMIC__PhyFullSpeed
Decimator gets one sample per each chosen clock edge of PDM interface

enumerator kDMIC__PhyHalfSpeed
PDM clock to Microphone is halved, decimator receives each sample twice

typedef enum *operation_mode* operation_mode_t

DMIC different operation modes.

typedef enum *stereo_side* stereo_side_t

DMIC left/right values.

`typedef enum _compensation compensation_t`
Pre-emphasis Filter coefficient value for 2FS and 4FS modes.

`typedef enum _dc_removal dc_removal_t`
DMIC DC filter control values.

`typedef enum _dmic_channel dmic_channel_t`
DMIC Channel number.

`typedef enum _dmic_phy_sample_rate dmic_phy_sample_rate_t`
DMIC and decimator sample rates.

`typedef struct _dmic_channel_config dmic_channel_config_t`
DMIC Channel configuration structure.

`typedef void (*dmic_callback_t)(void)`
DMIC Callback function.

`typedef void (*dmic_hwvad_callback_t)(void)`
HWVAD Callback function.

`struct _dmic_channel_config`
`#include <fsl_dmic.h>` DMIC Channel configuration structure.

Public Members

`pdm_div_t` divhfelk
DMIC Clock pre-divider values

`uint32_t` osr
oversampling rate(CIC decimation rate) for PCM

`uint32_t` gainshift
4FS PCM data gain control

`compensation_t` preac2coef
Pre-emphasis Filter coefficient value for 2FS

`compensation_t` preac4coef
Pre-emphasis Filter coefficient value for 4FS

`dc_removal_t` dc_cut_level
DMIC DC filter control values.

`uint32_t` post_dc_gain_reduce
Fine gain adjustment in the form of a number of bits to downshift

`dmic_phy_sample_rate_t` sample_rate
DMIC and decimator sample rates

`bool` saturate16bit
Selects 16-bit saturation. 0 means results roll over if out range and do not saturate. 1 means if the result overflows, it saturates at 0xFFFF for positive overflow and 0x8000 for negative overflow.

`bool` enableSignExtend
sign extend feature which allows processing of 24bit audio data on 32bit machine

2.18 Efuse_driver

status_t EFUSE_Init(*void*)

Initialize EFUSE controller.

This function enables EFUSE Controller clock.

Return values

- kStatus_Success – 0 Operation succeeded without error.
- kStatus_Fail – 1 The operation failed with a generic error.
- kStatus_ReadOnly – 2 Requested value cannot be changed because it is read-only.
- kStatus_OutOfRange – 3 Requested value is out of range.
- kStatus_InvalidArgument – 4 The requested command's argument is undefined.
- kStatus_Timeout – 5 A timeout occurred
- kStatus_NoTransferInProgress – 6 No send in progress.

status_t EFUSE_Deinit(*void*)

De-Initialize EFUSE controller.

This function disables EFUSE Controller Clock.

Return values

- kStatus_Success – 0 Operation succeeded without error.
- kStatus_Fail – 1 The operation failed with a generic error.
- kStatus_ReadOnly – 2 Requested value cannot be changed because it is read-only.
- kStatus_OutOfRange – 3 Requested value is out of range.
- kStatus_InvalidArgument – 4 The requested command's argument is undefined.
- kStatus_Timeout – 5 A timeout occurred
- kStatus_NoTransferInProgress – 6 No send in progress.

status_t EFUSE_Read(*uint32_t* addr, *uint32_t* *data)

Read Fuse value from eFuse word.

This function read fuse data from eFuse word to specified data buffer.

Parameters

- addr – Fuse address
- data – Buffer to hold the data read from eFuse word

Return values

- kStatus_Success – 0 Operation succeeded without error.
- kStatus_Fail – 1 The operation failed with a generic error.
- kStatus_ReadOnly – 2 Requested value cannot be changed because it is read-only.
- kStatus_OutOfRange – 3 Requested value is out of range.
- kStatus_InvalidArgument – 4 The requested command's argument is undefined.

- `kStatus_Timeout` – An invalid 5 A timeout occurred
- `kStatus_NoTransferInProgress` – 6 No send in progress.

`status_t` `EFUSE_Program(uint32_t addr, uint32_t data)`

Program value to eFuse block.

This function program data to specified eFuse address.

Parameters

- `addr` – Fuse address
- `data` – data to be programmed into eFuse Fuse block

Return values

- `kStatus_Success` – 0 Operation succeeded without error.
- `kStatus_Fail` – 1 The operation failed with a generic error.
- `kStatus_ReadOnly` – 2 Requested value cannot be changed because it is read-only.
- `kStatus_OutOfRange` – 3 Requested value is out of range.
- `kStatus_InvalidArgument` – 4 The requested command's argument is undefined.
- `kStatus_Timeout` – An invalid 5 A timeout occurred
- `kStatus_NoTransferInProgress` – 6 No send in progress.

2.19 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 inputs 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_FilteredPHASStatusFlag

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.

2.20 Flash_driver

enum _flash_driver_version_constants

Flash driver version for ROM.

Values:

enumerator kFLASH_DriverVersionName

Flash driver version name.

enumerator kFLASH_DriverVersionMajor

Major flash driver version.

enumerator kFLASH_DriverVersionMinor

Minor flash driver version.

enumerator kFLASH_DriverVersionBugfix

Bugfix for flash driver version.

FSL_ROMAPI_FLASH_DRIVER_VERSION

ROMAPI_FLASH driver version 2.0.1.

Flash driver status codes.

Values:

enumerator kStatus_FLASH_Success

API is executed successfully

enumerator kStatus_FLASH_InvalidArgument

Invalid argument

enumerator kStatus_FLASH_SizeError

Error size

enumerator kStatus_FLASH_AlignmentError

Parameter is not aligned with the specified baseline

enumerator kStatus_FLASH_AddressError

Address is out of range

enumerator kStatus_FLASH_AccessError

Invalid instruction codes and out-of bound addresses

enumerator kStatus_FLASH_ProtectionViolation

The program/erase operation is requested to execute on protected areas

enumerator kStatus_FLASH_CommandFailure

Run-time error during command execution.

enumerator kStatus_FLASH_UnknownProperty

Unknown property.

enumerator kStatus_FLASH_EraseKeyError

API erase key is invalid.

enumerator kStatus_FLASH_RegionExecuteOnly

The current region is execute-only.

enumerator kStatus_FLASH_ExecuteInRamFunctionNotReady

Execute-in-RAM function is not available.

enumerator kStatus_FLASH_CommandNotSupported

Flash API is not supported.

enumerator kStatus_FLASH_ReadOnlyProperty

The flash property is read-only.

enumerator kStatus_FLASH_InvalidPropertyValue

The flash property value is out of range.

enumerator kStatus_FLASH_InvalidSpeculationOption

The option of flash prefetch speculation is invalid.

enumerator kStatus_FLASH_EccError

A correctable or uncorrectable error during command execution.

enumerator kStatus_FLASH_CompareError

Destination and source memory contents do not match.

enumerator kStatus_FLASH_RegulationLoss

A loss of regulation during read.

enumerator kStatus_FLASH_InvalidWaitStateCycles

The wait state cycle set to r/w mode is invalid.

enumerator kStatus_FLASH_OutOfDateCfpaPage

CFPA page version is out of date.

enumerator kStatus_FLASH_BlankIfrPageData

Blank page cannot be read.

enumerator kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce

Encrypted flash subregions are not erased at once.

enumerator kStatus_FLASH_ProgramVerificationNotAllowed

Program verification is not allowed when the encryption is enabled.

enumerator kStatus_FLASH_HashCheckError

Hash check of page data is failed.

enumerator kStatus_FLASH_SealedFfrRegion

The FFR region is sealed.

enumerator kStatus_FLASH_FfrRegionWriteBroken

The FFR Spec region is not allowed to be written discontinuously.

enumerator kStatus_FLASH_NmpaAccessNotAllowed

The NMPA region is not allowed to be read/written/erased.

enumerator kStatus_FLASH_CmpaCfgDirectEraseNotAllowed

The CMPA Cfg region is not allowed to be erased directly.

enumerator kStatus_FLASH_FfrBankIsLocked

The FFR bank region is locked.

enumerator kStatus_FLASH_CfpaScratchPageInvalid

CFPA Scratch Page is invalid

enumerator kStatus_FLASH_CfpaVersionRollbackDisallowed

CFPA version rollback is not allowed

enumerator kStatus_FLASH_ReadHidingAreaDisallowed

Flash hiding read is not allowed

enumerator kStatus_FLASH_ModifyProtectedAreaDisallowed

Flash firewall page locked erase and program are not allowed

enumerator kStatus_FLASH_CommandOperationInProgress

The flash state is busy, indicate that a flash command in progress.

status_t FLASH_IsFlashAreaReadable(*flash_config_t* *config, uint32_t startAddress, uint32_t lengthInBytes)

Validates the given address range is loaded in the flash hiding region.

Parameters

- config – A pointer to the storage for the driver runtime state.
- startAddress – The start address of the desired flash memory to be verified.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed.

status_t FLASH_IsFlashAreaModifiable(*flash_config_t* *config, uint32_t startAddress, uint32_t lengthInBytes)

Validates the given address range is loaded in the Flash firewall page locked region.

Parameters

- config – A pointer to the storage for the driver runtime state.
- startAddress – The start address of the desired flash memory to be verified.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash hiding read is not allowed.

kStatusGroupGeneric

Flash driver status group.

kStatusGroupFlashDriver

MAKE_STATUS(group, code)

Constructs a status code value from a group and a code number.

enum _flash_driver_api_keys

Enumeration for Flash driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFLASH_ApiEraseKey

Key value used to validate all flash erase APIs.

enum _flash_property_tag

Enumeration for various flash properties.

Values:

enumerator kFLASH_PropertyPflashSectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflashTotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflashBlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflashBlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflashBlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflashPageSize
Pflash page size property.

enumerator kFLASH_PropertyPflashSystemFreq
System Frequency System Frequency.

enumerator kFLASH_PropertyFfrSectorSize
FFR sector size property.

enumerator kFLASH_PropertyFfrTotalSize
FFR total size property.

enumerator kFLASH_PropertyFfrBlockBaseAddr
FFR block base address property.

enumerator kFLASH_PropertyFfrPageSize
FFR page size property.

enum _flash_max_erase_page_value

Enumeration for flash max pages to erase.

Values:

enumerator kFLASH_MaxPagesToErase
The max value in pages to erase.

enum _flash_alignment_property

Enumeration for flash alignment property.

Values:

enumerator kFLASH_AlignementUnitVerifyErase
The alignment unit in bytes used for verify erase operation.

enumerator kFLASH_AlignementUnitProgram
The alignment unit in bytes used for program operation.

enumerator kFLASH_AlignementUnitSingleWordRead
The alignment unit in bytes used for verify program operation. The alignment unit in bytes used for SingleWordRead command.

enum _flash_read_ecc_option

Enumeration for flash read ecc option.

Values:

enumerator kFLASH_ReadWithEccOn

enumerator kFLASH_ReadWithEccOff

ECC is on

enum _flash_read_margin_option

Enumeration for flash read margin option.

Values:

enumerator kFLASH_ReadMarginNormal

Normal read

enumerator kFLASH_ReadMarginVsProgram

Margin vs. program

enumerator kFLASH_ReadMarginVsErase

Margin vs. erase

enumerator kFLASH_ReadMarginIllegalBitCombination

Illegal bit combination

enum _flash_read_dmacc_option

Enumeration for flash read dmacc option.

Values:

enumerator kFLASH_ReadDmaccDisabled

Memory word

enumerator kFLASH_ReadDmaccEnabled

DMACC word

enum _flash_ramp_control_option

Enumeration for flash ramp control option.

Values:

enumerator kFLASH_RampControlDivisionFactorReserved

Reserved

enumerator kFLASH_RampControlDivisionFactor256

$\text{clk48mhz} / 256 = 187.5\text{KHz}$

enumerator kFLASH_RampControlDivisionFactor128

$\text{clk48mhz} / 128 = 375\text{KHz}$

enumerator kFLASH_RampControlDivisionFactor64

$\text{clk48mhz} / 64 = 750\text{KHz}$

typedef enum _flash_property_tag flash_property_tag_t

Enumeration for various flash properties.

typedef struct _flash_ecc_log flash_ecc_log_t

Flash ECC log info.

typedef struct _flash_mode_config flash_mode_config_t

Flash controller paramter config.

typedef struct _flash_ffr_config flash_ffr_config_t

Flash controller paramter config.

FOUR_CHAR_CODE(a, b, c, d)

Constructs the four character code for the Flash driver API key.

status_t FLASH_Init(*flash_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FLASH_Erase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. NOTE: The start address need to be 4 Bytes-aligned.
- lengthInBytes – The length, given in bytes need be 4 Bytes-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FLASH_AddressError – The address is out of range.
- kStatus_FLASH_EraseKeyError – The API erase key is invalid.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FLASH_EraseNonBlocking(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Non-blocking Erases the flash sectors encompassed by parameters passed into function.

This is a non-blocking function, which returns right away. This function erases the appropriate number of flash sectors based on the desired start address and length, and get the command execute status from the “FLASH_GetCommandState”.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. NOTE: The start address need be 4 Bytes-aligned.
- lengthInBytes – The length, given in bytes need be 4 Bytes-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FLASH_AddressError – The address is out of range.
- kStatus_FLASH_EraseKeyError – The API erase key is invalid.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed

status_t FLASH_Program(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_AddressError – Address is out of range.
- kStatus_FLASH_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed

- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t FLASH_Read(flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t lengthInBytes)`

Reads flash at locations passed in through parameters.

This function read the flash memory from a given flash area as determined by the start address and the length.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be read.
- `dest` – A pointer to the dest buffer of data that is to be read from the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be read.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_ReadHidingAreaDisallowed` – Flash hiding read is not allowed
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t FLASH_VerifyErase(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)`

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.

- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t FLASH_VerifyProgram(flash_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, uint32_t *failedAddress, uint32_t *failedData)`

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `expectedData` – A pointer to the expected data that is to be verified against.
- `failedAddress` – A pointer to the returned failing address.
- `failedData` – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FLASH_ReadHidingAreaDisallowed` – Flash hiding read is not allowed
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

status_t FLASH_GetProperty(*flash_config_t* *config, *flash_property_tag_t* whichProperty, *uint32_t* *value)

Returns the desired flash property.

Parameters

- config – A pointer to the storage for the driver runtime state.
- whichProperty – The desired property from the list of properties in enum *flash_property_tag_t*
- value – A pointer to the value returned for the desired flash property.

Return values

- *kStatus_FLASH_Success* – API was executed successfully.
- *kStatus_FLASH_InvalidArgument* – An invalid argument is provided.
- *kStatus_FLASH_UnknownProperty* – An unknown property tag.

status_t FLASH_GetCommandState(*flash_config_t* *config)

Get flash command execute status.

This function is used to obtain the status after the command “FLASH_EraseNonBlocking” is executed.

Parameters

- config – A pointer to the storage for the driver runtime state.

Return values

- *kStatus_FLASH_CommandOperationInProgress* – Indicate that a flash command in progress.
- *kStatus_FLASH_CommandFailure* – Run-time error during the command execution.
- *kStatus_FLASH_CommandNotSupported* – Flash API is not supported.
- *kStatus_FLASH_EccError* – A correctable or uncorrectable error during command execution.
- *kStatus_FLASH_RegulationLoss* – A loss of regulation during read.
- *kStatus_FLASH_Success* – API was executed successfully.

MAKE_VERSION(major, minor, bugfix)

Constructs the version number for drivers.

FSL_SUPPORT_ERASE_SECTOR_NON_BLOCKING

IAP driver support non-block erase function.

FSL_FEATURE_SYSCON_HAS_FLASH_HIDING

IAP driver support flash hiding.

FSL_FEATURE_SYSCON_HAS_CDPA

IAP driver support CDPA.

uint32_t firstEccEventAddress

uint32_t eccErrorCount

uint32_t eccCorrectionCount

uint32_t reserved

uint32_t sysFreqInMHz

```
uint8_t readWithEccOff
uint8_t readMarginLevel
uint8_t readDmaccWord
uint8_t reserved0
uint8_t reserved1[3]
struct _flash_mode_config readSingleWord
uint8_t programRampControl
uint8_t eraseRampControl
uint8_t reserved[2]
struct _flash_mode_config setWriteMode
uint16_t readInterfaceTimingTrim
uint16_t readControllerTimingTrim
uint8_t readWaitStates
uint8_t reserved[3]
struct _flash_mode_config setReadMode
uint32_t ffrBlockBase
uint32_t ffrTotalSize
uint32_t ffrPageSize
uint32_t cfpaPageVersion
uint32_t cfpaPageOffset
uint32_t PFlashBlockBase
    A base address of the first PFlash block
uint32_t PFlashTotalSize
    The size of the combined PFlash block.
uint32_t PFlashBlockCount
    A number of PFlash blocks.
uint32_t PFlashPageSize
    The size in bytes of a page of PFlash.
uint32_t PFlashSectorSize
    The size in bytes of a sector of PFlash.
flash_ffr_config_t ffrConfig
flash_mode_config_t modeConfig
uint32_t *nbootCtx
bool useAhbRead
struct _flash_ecc_log
    #include <fsl_flash.h> Flash ECC log info.
```



```
struct _flash_mode_config
```

#include <fsl_flash.h> Flash controller paramter config.

```
struct _flash_ffr_config
```

#include <fsl_flash.h> Flash controller paramter config.

```
struct flash_config_t
```

#include <fsl_flash.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

```
struct readSingleWord
```

```
struct setWriteMode
```

```
struct setReadMode
```

2.21 Flash_ffr_driver

```
status_t FFR_Init(flash_config_t *config)
```

Initializes the global FFR properties structure members.

Parameters

- config – A pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success – API was executed successfully.

```
status_t FFR_Lock(flash_config_t *config)
```

Enable firewall for all flash banks.

CFPA, CMPA, and NMPA flash areas region will be locked, After this function executed; Unless the board is reset again.

Parameters

- config – A pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success – An invalid argument is provided.

```
status_t FFR_SecLibInit(flash_config_t *config, uint32_t *context)
```

Initialize the Security Library for FFR driver.

Parameters

- config – A pointer to the storage for the driver runtime state.
- context – A pointer to the storage for the nboot data.

Return values

- kStatus_FLASH_Success – An invalid argument is provided.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.

```
status_t FFR_InfieldPageWrite(flash_config_t *config, uint8_t *page_data, uint32_t valid_len)
```

APIs to access CFPA pages.

This routine will erase CFPA and program the CFPA page with passed data.

Parameters

- config – A pointer to the storage for the driver runtime state.

- `page_data` – A pointer to the source buffer of data that is to be programmed into the CFPA.
- `valid_len` – The length, given in bytes, to be programmed.

Return values

- `kStatus_FLASH_Success` – The desire page-data were programed successfully into CFPA.
- `kStatus_FLASH_SizeError` – Error size
- `kStatus_FLASH_ReadHidingAreaDisallowed` – Flash hiding read is not allowed
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` – Flash firewall page locked erase and program are not allowed
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_FfrBankIsLocked` – The CFPA was locked.
- `kStatus_FLASH_OutOfDateCfpaPage` – It is not newest CFPA page.
- `kStatus_FLASH_CommandFailure` – access error.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.

`status_t` FFR_GetCustomerInfieldData(*flash_config_t* *config, `uint8_t` *pData, `uint32_t` offset, `uint32_t` len)

APIs to access CFPA pages.

Generic read function, used by customer to read data stored in ‘Customer In-field Page’.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `pData` – A pointer to the dest buffer of data that is to be read from ‘Customer In-field Page’.
- `offset` – An offset from the ‘Customer In-field Page’ start address.
- `len` – The length, given in bytes, to be read.

Return values

- `kStatus_FLASH_Success` – Get data from ‘Customer In-field Page’.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_ReadHidingAreaDisallowed` – Flash hiding read is not allowed
- `kStatus_FLASH_CommandFailure` – access error.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported

- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.

`status_t` FFR_CustFactoryPageWrite(*flash_config_t* *config, *uint8_t* *page_data, *bool* seal_part)
APIs to access CMPA pages.

This routine will erase “customer factory page” and program the page with passed data. If ‘seal_part’ parameter is TRUE then the routine will compute SHA256 hash of the page contents and then programs the pages. 1.During development customer code uses this API with ‘seal_part’ set to FALSE. 2.During manufacturing this parameter should be set to TRUE to seal the part from further modifications 3.This routine checks if the page is sealed or not. A page is said to be sealed if the SHA256 value in the page has non-zero value. On boot ROM locks the firewall for the region if hash is programmed anyways. So, write/erase commands will fail eventually.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `page_data` – A pointer to the source buffer of data that is to be programmed into the “customer factory page”.
- `seal_part` – Set false for During development customer code.

Return values

- `kStatus_FLASH_Success` – The desire page-data were programed successfully into CMPA.
- `kStatus_FLASH_InvalidArgument` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_EraseKeyError` – API erase key is invalid.
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` – Flash firewall page locked erase and program are not allowed
- `kStatus_Fail` – Generic status for Fail.
- `kStatus_FLASH_CommandFailure` – access error.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.

`status_t` FFR_GetCustomerData(*flash_config_t* *config, *uint8_t* *pData, *uint32_t* offset, *uint32_t* len)

APIs to access CMPA page.

Read data stored in ‘Customer Factory CFG Page’.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `pData` – A pointer to the dest buffer of data that is to be read from the Customer Factory CFG Page.
- `offset` – Address offset relative to the CMPA area.
- `len` – The length, given in bytes to be read.

Return values

- `kStatus_FLASH_Success` – Get data from ‘Customer Factory CFG Page’.
- `kStatus_FLASH_InvalidArgument` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_CommandFailure` – access error.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_ReadHidingAreaDisallowed` – Flash hiding read is not allowed

`status_t` FFR_GetCustKeystoreData(*flash_config_t* *config, `uint8_t` *pData, `uint32_t` offset, `uint32_t` len)

The API is used for getting the customer key store data from the customer key store region(0x3e400 - 0x3e600), and the API should be called after the FLASH_Init and FFR_Init.

Parameters

- config – A pointer to the storage for the driver runtime state.
- pData – A pointer to the dest buffer of data that is to be read from the Customer Factory CFG Page.
- offset – Address offset relative to the CMPA area.
- len – The length, given in bytes to be read.

Return values

- `kStatus_FLASH_Success` – Get data from ‘Customer Factory CFG Page’.
- `kStatus_FLASH_InvalidArgument` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_ReadHidingAreaDisallowed` – Flash hiding read is not allowed
- `kStatus_FLASH_CommandFailure` – access error.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.

`status_t` FFR_CustKeystoreWrite(*flash_config_t* *config, *ffr_key_store_t* *pKeyStore)

This routine writes the 3 pages allocated for Key store data.

Parameters

- config – A pointer to the storage for the driver runtime state.

- pKeyStore – A pointer to the source buffer of data that is to be programmed into the “Key store”.

Return values

- kStatus_FLASH_Success – Get data from ‘Customer Factory CFG Page’.
- kStatus_FLASH_InvalidArgument – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_CommandFailure – access error.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_SealedFfrRegion – The FFR region is sealed.
- kStatus_FLASH_FfrBankIsLocked – The FFR bank region is locked.
- kStatus_FLASH_AddressError – Address is out of range
- kStatus_FLASH_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_ModifyProtectedAreaDisallowed – Flash firewall page locked erase and program are not allowed

status_t FFR_GetUUID(*flash_config_t* *config, *uint8_t* *uuid)

APIs to access CMPA page.

1.SW should use this API routine to get the UUID of the chip. 2.Calling routine should pass a pointer to buffer which can hold 128-bit value.

Return values

- kStatus_FLASH_Success – Get data from ‘Customer Factory CFG Page’.
- kStatus_FLASH_InvalidArgument – Parameter is not aligned with the specified baseline.
- kStatus_FLASH_ReadHidingAreaDisallowed – Flash hiding read is not allowed
- kStatus_FLASH_CommandFailure – Run-time error during command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

enum flash_ffr_page_offset

flash FFR page offset.

Values:

enumerator kFfrPageOffset_CFPA

Customer In-Field programmed area

enumerator kFfrPageOffset_CFPA_Scratch

CFPA Scratch page

enumerator kFfrPageOffset_CFPA_CfgPing

CFPA Configuration area (Ping page)

enumerator kFfrPageOffset_CFPA_CfgPong

Same as CFPA page (Pong page)

enumerator kFfrPageOffset__CMPA
Customer Manufacturing programmed area

enumerator kFfrPageOffset__CMPA__Cfg
CMPA Configuration area (Part of CMPA)

enumerator kFfrPageOffset__CMPA__Key
Key Store area (Part of CMPA)

enumerator kFfrPageOffset__NMPA
NXP Manufacturing programmed area

enumerator kFfrPageOffset__NMPA__Romcp
ROM patch area (Part of NMPA)

enumerator kFfrPageOffset__NMPA__Repair
Repair area (Part of NMPA)

enumerator kFfrPageOffset__NMPA__Cfg
NMPA configuration area (Part of NMPA)

enumerator kFfrPageOffset__NMPA__End
Reserved (Part of NMPA)

enum flash_ffr_page_num
flash FFR page number.

Values:

enumerator kFfrPageNum__CFPA
Customer In-Field programmed area

enumerator kFfrPageNum__CMPA
Customer Manufacturing programmed area

enumerator kFfrPageNum__NMPA
NXP Manufacturing programmed area

enumerator kFfrPageNum__CMPA__Cfg

enumerator kFfrPageNum__NMPA__Romcp

enumerator kFfrPageNum__SpecArea

enumerator kFfrPageNum__Total

enum flash_ffr_block_size
flash FFR block size.

Values:

enumerator kFfrBlockSize__Key

enumerator kFfrBlockSize__ActivationCode

enum cfp_a_cfg_cmpa_prog_status
flash cmpa status.

Values:

enumerator kFfrCmpaProgStatus__Idle

enumerator kFfrCmpaProgStatus__InProgress

enum cmpa_prog_process_t

flash cmpa program status.

Values:

enumerator kFfrCmpaProgProcess_Pre

enumerator kFfrCmpaProgProcess_Post

enum ffr_key_type_t

Values:

enumerator kFFR_KeyTypeSbkek

enumerator kFFR_KeyTypeUser

enumerator kFFR_KeyTypeUds

enumerator kFFR_KeyTypePrinceRegion0

enumerator kFFR_KeyTypePrinceRegion1

enumerator kFFR_KeyTypePrinceRegion2

enum ffr_bank_type_t

Values:

enumerator kFFR_BankTypeBank0_NMPA

enumerator kFFR_BankTypeBank1_CMPA

enumerator kFFR_BankTypeBank2_CFPA

uint32_t header

[0x000-0x003]

uint32_t version

[0x004-0x007]

uint32_t secureFwVersion

[0x008-0x00b]

uint32_t nsFwVersion

[0x00c-0x00f]

uint32_t imageKeyRevoke

[0x010-0x013]

uint32_t ivPrince[3]

[0x014-0x01f]

uint32_t ivIped[4]

[0x020-0x02f]

uint32_t custCtr[8]

[0x030-0x03f]

uint32_t rotkhRevoke

[0x018-0x01b]

uint32_t vendorUsage

[0x050-0x053]

uint32_t dcfgNsPin
[0x058-0x05b]

uint32_t dcfgNsDflt
[0x05c-0x05f]

uint32_t enableFaMode
[0x060-0x063]

uint32_t cmpaProgInProgress
[0x064-0x067]

uint32_t imageCmacUpdateEn
[0x068-0x06b]

uint32_t cfpaVersion
[0x06c-0x06f]

uint32_t img0Cmac[4]
[0x070-0x07f]

uint32_t img1Cmac[4]
[0x080-0x08f]

uint8_t reserved2[348]
[0x090-0x1eb]

uint32_t cfpaCrc
[0x1eb-0x1ef]

uint32_t cfpaCmac[4]
[0x1f0-0x1ff]

uint32_t bootCfg
[0x000-0x003]

uint32_t spiFlashCanCfg
[0x004-0x007]

uint16_t vid

uint16_t pid

struct *cmpa_cfg_info_t* usbId
[0x008-0x00b]

uint32_t sdioCfg
[0x00c-0x00f]

uint32_t dcfgPin
[0x010-0x013]

uint32_t dcfgDflt
[0x014-0x017]

uint32_t dapVendorUsage
[0x018-0x01b]

uint32_t secureBootCfg
[0x01c-0x01f]

uint32_t princeBaseAddr
[0x020-0x023]


```
uint32_t princeSr[3]
    [0x024-0x02f]
uint32_t xtal32kCapBankTrim
    [0x030-0x033]
uint32_t xtal16mCapBankTrim
    [0x034-0x037]
uint32_t flashRemapSize
    [0x038-0x03b]
uint32_t flashRemapOffset
    [0x03c-0x03f]
uint32_t princeXom[3]
    [0x040-0x04b]
uint32_t rokthUsage
    [0x04c-0x04f]
uint32_t rotkh[12]
    [0x050-0x07f]
uint32_t flexspiCfg0
    [0x080-0x083]
uint32_t flexspiCfg1
    [0x084-0x087]
uint8_t reserved1[8]
    [0x088-0x08f]
uint32_t ipedStartAddr
    [0x090-0x093]
uint32_t ipedEndAddr
    [0x094-0x097]
struct empa_cfg_info_t ipedRegions[4]
uint8_t reserved2[320]
    [0x0b0-0x1ef]
uint32_t empaCmac[4]
    [0x1f0-0x1ff]
uint32_t header
uint8_t reserved[4]
uint32_t fro32kCfg
    [0x000-0x003]
uint32_t puf_cfg
    [0x004-0x007]
uint32_t bod
    [0x008-0x00b]
uint32_t trim
    [0x00c-0x00f]
```

```
uint32_t deviceID
    [0x010-0x03f]
uint32_t peripheralCfg
    [0x014-0x017]
uint32_t dcdPowerProFileLOW[2]
    [0x018-0x01f]
uint32_t deviceType
    [0x020-0x023]
uint32_t ldo_ao
    [0x024-0x027]
uint32_t gdetDelayCfg
    [0x028-0x02b]
uint32_t gdetMargin
    [0x02c-0x02f]
uint32_t gdetTrim1
    [0x030-0x033]
uint32_t gdetEanble1
    [0x034-0x037]
uint32_t gdetCtrl1
    [0x038-0x03b]
uint32_t gdetUpdateTimer
    [0x03c-0x03f]
uint32_t GpoDataChecksum[4]
    [0x040-0x04f]
uint32_t finalTestBatchId[4]
    [0x050-0x05f]
uint32_t ecidBackup[4]
    [0x060-0x06f]
uint32_t uuid[4]
    [0x070-0x07f]
uint32_t reserved1[7]
    [0x080-0x09b]
uint8_t xo32mReadyTimeoutInMs
uint8_t usbSpeed
uint8_t reserved[2]
struct nmpa_cfg_info_t usbCfg
    [0x09c-0x09f]
uint32_t reserved2[80]
    [0x0a0-0x1df]
uint8_t cmac[16]
    [0x1e0-0x1ef]
```

uint32_t pageChecksum[4]
 [0x1f0-0x1ff]
uint8_t reserved[1][(512U)]
ALIGN_DOWN(x, a)
 Alignment(down) utility.
ALIGN_UP(x, a)
 Alignment(up) utility.
FLASH_FFR_MAX_PAGE_SIZE
FLASH_FFR_HASH_DIGEST_SIZE
FLASH_FFR_IV_CODE_SIZE
FFR_BOOTCFG_USBSPEED_SHIFT
FFR_BOOTCFG_USBSPEED_MASK
FFR_BOOTCFG_USBSPEED_NMPASEL0
FFR_BOOTCFG_USBSPEED_FS
FFR_BOOTCFG_USBSPEED_HS
FFR_BOOTCFG_USBSPEED_NMPASEL3
FFR_BOOTCFG_BOOTSPEED_MASK
FFR_BOOTCFG_BOOTSPEED_SHIFT
FFR_BOOTCFG_BOOTSPEED_NMPASEL
FFR_BOOTCFG_BOOTSPEED_48MHZ
FFR_BOOTCFG_BOOTSPEED_96MHZ
FFR_USBID_VENDORID_MASK
FFR_USBID_VENDORID_SHIFT
FFR_USBID_PRODUCTID_MASK
FFR_USBID_PRODUCTID_SHIFT
FFR_IMAGE0_CMACE_UPDATE_MASK
FFR_IMAGE1_CMACE_UPDATE_MASK
FFR_SYSTEM_SPEED_CODE_MASK
FFR_SYSTEM_SPEED_CODE_SHIFT
FFR_SYSTEM_SPEED_CODE_FRO12MHZ_12MHZ
FFR_SYSTEM_SPEED_CODE_FROHF96MHZ_24MHZ
FFR_SYSTEM_SPEED_CODE_FROHF96MHZ_48MHZ
FFR_SYSTEM_SPEED_CODE_FROHF96MHZ_96MHZ
FFR_USBCFG_USBSPEED_HS

FFR_USBCFG_USBSPEED_FS
FFR_USBCFG_USBSPEED_NO
FFR_MCAN_BAUDRATE_MASK
FFR_MCAN_BAUDRATE_SHIFT
FFR_PERIPHERALCFG_PERI_MASK
FFR_PERIPHERALCFG_PERI_SHIFT
FFR_PERIPHERALCFG_COREEN_MASK
FFR_PERIPHERALCFG_COREEN_SHIFT
FFR_PUF_SRAM_CONFIG_MASK
FFR_PUF_SRAM_CONFIG_MASK_SHIFT
FFR_PUF_SRAM_VALID_MASK
FFR_PUF_SRAM_VALID_SHIFT
FFR_PUF_SRAM_MODE_MASK
FFR_PUF_SRAM_MODE_SHIFT
FFR_PUF_SRAM_CKGATING_MASK
FFR_PUF_SRAM_CKGATING_SHIFT
FFR_PUF_SRAM_SMB_MASK
FFR_PUF_SRAM_SMB_SHIFT
FFR_PUF_SRAM_RM_MASK
FFR_PUF_SRAM_RM_SHIFT
FFR_PUF_SRAM_WM_MASK
FFR_PUF_SRAM_WM_SHIFT
FFR_PUF_SRAM_WRME_MASK
FFR_PUF_SRAM_WRME_SHIFT
FFR_PUF_SRAM_RAEN_MASK
FFR_PUF_SRAM_RAEN_SHIFT
FFR_PUF_SRAM_RAM_MASK
FFR_PUF_SRAM_RAM_SHIFT
FFR_PUF_SRAM_WAEN_MASK
FFR_PUF_SRAM_WAEN_SHIFT
FFR_PUF_SRAM_WAM_MASK
FFR_PUF_SRAM_WAM_SHIFT
FFR_PUF_SRAM_STBP_MASK

FFR_PUF_SRAM_STBP_SHIFT

struct cfpa_cfg_info_t
 #include <fsl_flash_ffr.h> flash cfpa configure struct.

struct cmpa_cfg_info_t
 #include <fsl_flash_ffr.h>

struct cmpa_key_store_header_t
 #include <fsl_flash_ffr.h>

struct nmpa_cfg_info_t
 #include <fsl_flash_ffr.h>

struct ffr_key_store_t
 #include <fsl_flash_ffr.h>

struct usbId

struct ipedRegions

Public Members

uint32_t ipedStartAddr
 [0x090-0x093]

uint32_t ipedEndAddr
 [0x094-0x097]

struct usbCfg

2.22 FLEXCOMM: FLEXCOMM Driver

2.23 FLEXCOMM Driver

FSL_FLEXCOMM_DRIVER_VERSION
FlexCOMM driver version 2.0.2.

enum FLEXCOMM_PERIPH_T
FlexCOMM peripheral modes.

Values:

enumerator FLEXCOMM_PERIPH_NONE
No peripheral

enumerator FLEXCOMM_PERIPH_USART
USART peripheral

enumerator FLEXCOMM_PERIPH_SPI
SPI Peripheral

enumerator FLEXCOMM_PERIPH_I2C
I2C Peripheral

enumerator FLEXCOMM_PERIPH_I2S_TX
I2S TX Peripheral

enumerator FLEXCOMM_PERIPH_I2S_RX

I2S RX Peripheral

typedef void (*flexcomm_irq_handler_t)(void *base, void *handle)

Typedef for interrupt handler.

IRQn_Type const kFlexcommIrqs[]

Array with IRQ number for each FLEXCOMM module.

uint32_t FLEXCOMM_GetInstance(void *base)

Returns instance number for FLEXCOMM module with given base address.

status_t FLEXCOMM_Init(void *base, FLEXCOMM_PERIPH_T periph)

Initializes FLEXCOMM and selects peripheral mode according to the second parameter.

void FLEXCOMM_SetIRQHandler(void *base, flexcomm_irq_handler_t handler, void *flexcommHandle)

Sets IRQ handler for given FLEXCOMM module. It is used by drivers register IRQ handler according to FLEXCOMM mode.

2.24 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32_t FLEXSPI_GetInstance(FLEXSPI_Type *base)

Get the instance number for FLEXSPI.

Parameters

- base – FLEXSPI base pointer.

status_t FLEXSPI_CheckAndClearError(FLEXSPI_Type *base, uint32_t status)

Check and clear IP command execution errors.

Parameters

- base – FLEXSPI base pointer.
- status – interrupt status.

void FLEXSPI_Init(FLEXSPI_Type *base, const flexspi_config_t *config)

Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI_GetDefaultConfig(flexspi_config_t *config)

Gets default settings for FLEXSPI.

Parameters

- config – FLEXSPI configuration structure.

void FLEXSPI_Deinit(FLEXSPI_Type *base)

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

- base – FLEXSPI peripheral base address.

```
void FLEXSPI_UpdateDllValue(FLEXSPI_Type *base, flexspi_device_config_t *config,  
                           flexspi_port_t port)
```

Update FLEXSPI DLL value depending on currently flexspi root clock.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

```
void FLEXSPI_SetFlashConfig(FLEXSPI_Type *base, flexspi_device_config_t *config,  
                           flexspi_port_t port)
```

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

```
void FLEXSPI_SoftwareReset(FLEXSPI_Type *base)
```

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – FLEXSPI peripheral base address.

```
static inline void FLEXSPI_Enable(FLEXSPI_Type *base, bool enable)
```

Enables or disables the FLEXSPI module.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

```
void FLEXSPI_UpdateAhbBuffersSettings(FLEXSPI_Type *base, flexspi_ahbBuffers_ctrl_t  
                                     *ptrAhbBufferCtrl)
```

Update all AHB buffers' settings, including buffer size, master ID.

Parameters

- base – FLEXSPI peripheral base address.
- ptrAhbBufferCtrl – Pointer to structure flexspi_ahbBuffers_ctrl_t which store all AHB buffers' settings.

```
static inline void FLEXSPI_EnableInterrupts(FLEXSPI_Type *base, uint32_t mask)
```

Enables the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)
```

Disable the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)
```

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

```
static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)
```

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

```
static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)
```

Gets FLEXSPI IP tx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – tx fifo address.

```
static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)
```

Gets FLEXSPI IP rx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – rx fifo address.

```
static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)
```

Clears the FLEXSPI IP FIFO logic.

Parameters

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

```
static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t  
                                         *rxCount)
```

Gets the valid data entries in the FLEXSPI FIFOs.

Parameters

- base – FLEXSPI peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.

- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)

Get the FLEXSPI interrupt status flags.

Parameters

- `base` – FLEXSPI peripheral base address.

Return values

`interrupt` – status flag, use status flag to AND `flexspi_flags_t` could get the related status.

static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)

Get the FLEXSPI interrupt status flags.

Parameters

- `base` – FLEXSPI peripheral base address.
- `mask` – FLEXSPI interrupt source.

static inline void FLEXSPI_GetDataLearningPhase(FLEXSPI_Type *base, uint8_t *portAPhase, uint8_t *portBPhase)

Gets the sampling clock phase selection after Data Learning.

Parameters

- `base` – FLEXSPI peripheral base address.
- `portAPhase` – Pointer to a `uint8_t` type variable to receive the selected clock phase on PORTA.
- `portBPhase` – Pointer to a `uint8_t` type variable to receive the selected clock phase on PORTB.

static inline *flexspi_arb_command_source_t* FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type *base)

Gets the trigger source of current command sequence granted by arbitrator.

Parameters

- `base` – FLEXSPI peripheral base address.

Return values

`trigger` – source of current command sequence.

static inline *flexspi_ip_error_code_t* FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when IP command error detected.

Parameters

- `base` – FLEXSPI peripheral base address.
- `index` – Pointer to a `uint8_t` type variable to receive the sequence index when error detected.

Return values

`error` – code when IP command error detected.

static inline *flexspi_ahb_error_code_t* FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when AHB command error detected.

Parameters

- `base` – FLEXSPI peripheral base address.
- `index` – Pointer to a `uint8_t` type variable to receive the sequence index when error detected.

Return values

`error` – code when AHB command error detected.

static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)

Returns whether the bus is idle.

Parameters

- `base` – FLEXSPI peripheral base address.

Return values

- `true` – Bus is idle.
- `false` – Bus is busy.

void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, flexspi_read_sample_clock_t clockSource)

Update read sample clock source.

Parameters

- `base` – FLEXSPI peripheral base address.
- `clockSource` – clockSource of type `flexspi_read_sample_clock_t`

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_DataLearningFailedFlag

Data learning failed.

enumerator kFLEXSPI_IpTxFifoWatermarkEmptyFlag

IP TX FIFO WaterMark empty.

enumerator kFLEXSPI_IpRxFifoWatermarkAvailableFlag

IP RX FIFO WaterMark available.

enumerator kFLEXSPI_AhbCommandSequenceErrorFlag

AHB triggered Command Sequences Error.

enumerator kFLEXSPI_IpCommandSequenceErrorFlag

IP triggered Command Sequences Error.

enumerator kFLEXSPI_AhbCommandGrantTimeoutFlag

AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandGrantTimeoutFlag

IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandExecutionDoneFlag

IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI_AllInterruptFlags

All flags.

enum _flexspi_read_sample_clock

FLEXSPI sample clock source selection for Flash Reading.

Values:

enumerator kFLEXSPI_ReadSampleClkLoopbackInternally

Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromDqsPad

Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromSckPad

SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI_ReadSampleClkExternalInputFromDqsPad

Flash provided Read strobe and input from DQS pad.

enum _flexspi_cs_interval_cycle_unit

FLEXSPI interval unit for flash device select.

Values:

enumerator kFLEXSPI_CsIntervalUnit1SckCycle

Chip selection interval: CSINTERVAL * 1 serial clock cycle.

enumerator kFLEXSPI_CsIntervalUnit256SckCycle

Chip selection interval: CSINTERVAL * 256 serial clock cycle.

enum _flexspi_ahb_write_wait_unit

FLEXSPI AHB wait interval unit for writing.

Values:

enumerator kFLEXSPI_AhbWriteWaitUnit2AhbCycle

AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8AhbCycle

AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32AhbCycle

AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit128AhbCycle

AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit512AhbCycle

AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit2048AhbCycle

AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8192AhbCycle

AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32768AhbCycle

AWRWAIT unit is 32768 ahb clock cycle.

enum _flexspi_ip_error_code

Error Code when IP command Error detected.

Values:

enumerator kFLEXSPI_IpCmdErrorNoError

No error.

enumerator kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd

IP command with JMP_ON_CS instruction used.

enumerator kFLEXSPI_IpCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_IpCmdErrorInvalidAddress

Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI_IpCmdErrorSequenceExecutionTimeout

Sequence execution timeout.

enumerator kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss

Flash boundary crossed.

enum _flexspi_ahb_error_code

Error Code when AHB command Error detected.

Values:

enumerator kFLEXSPI_AhbCmdErrorNoError

No error.

enumerator kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd

AHB Write command with JMP_ON_CS instruction used in the sequence.

enumerator kFLEXSPI_AhbCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_AhbCmdSequenceExecutionTimeout

Sequence execution timeout.

enum _flexspi_port

FLEXSPI operation port select.

Values:

enumerator kFLEXSPI_PortA1

Access flash on A1 port.

enumerator kFLEXSPI_PortA2

Access flash on A2 port.

enumerator kFLEXSPI_PortCount

enum _flexspi_arb_command_source

Trigger source of current command sequence granted by arbitrator.

Values:

enumerator kFLEXSPI_AhbReadCommand

enumerator kFLEXSPI_AhbWriteCommand

enumerator kFLEXSPI_IpCommand

enumerator kFLEXSPI_SuspendedCommand

enum *_flexspi_command_type*

Command type.

Values:

enumerator kFLEXSPI_Command

FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI_Config

FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI_Read

enumerator kFLEXSPI_Write

typedef enum *_flexspi_pad* flexspi_pad_t

pad definition of FLEXSPI, use to form LUT instruction.

typedef enum *_flexspi_flags* flexspi_flags_t

FLEXSPI interrupt status flags.

typedef enum *_flexspi_read_sample_clock* flexspi_read_sample_clock_t

FLEXSPI sample clock source selection for Flash Reading.

typedef enum *_flexspi_cs_interval_cycle_unit* flexspi_cs_interval_cycle_unit_t

FLEXSPI interval unit for flash device select.

typedef enum *_flexspi_ahb_write_wait_unit* flexspi_ahb_write_wait_unit_t

FLEXSPI AHB wait interval unit for writing.

typedef enum *_flexspi_ip_error_code* flexspi_ip_error_code_t

Error Code when IP command Error detected.

typedef enum *_flexspi_ahb_error_code* flexspi_ahb_error_code_t

Error Code when AHB command Error detected.

typedef enum *_flexspi_port* flexspi_port_t

FLEXSPI operation port select.

typedef enum *_flexspi_arb_command_source* flexspi_arb_command_source_t

Trigger source of current command sequence granted by arbitrator.

typedef enum *_flexspi_command_type* flexspi_command_type_t

Command type.

typedef struct *_flexspi_ahbBuffer_config* flexspi_ahbBuffer_config_t

typedef struct *_flexspi_ahbBuffers_ctrl* flexspi_ahbBuffers_ctrl_t

Structure to control all AHB buffers.

typedef struct *_flexspi_config* flexspi_config_t

FLEXSPI configuration structure.

typedef struct *_flexspi_device_config* flexspi_device_config_t

External device configuration items.

```
typedef struct _flexspi_transfer flexspi__transfer__t
```

Transfer structure for FLEXSPI.

```
typedef struct _flexspi_handle flexspi__handle__t
```

```
typedef void (*flexspi__transfer_callback__t)(FLEXSPI_Type *base, flexspi_handle_t *handle,  
status_t status, void *userData)
```

FLEXSPI transfer callback function.

```
typedef struct _flexspi_addr_map_config flexspi__addr_map_config__t
```

Address mapping configuration structure.

```
FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT
```

```
FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
```

Formula to form FLEXSPI instructions in LUT table.

```
struct _flexspi_ahbBuffer_config
```

```
#include <fsl_flexspi.h>
```

Public Members

```
uint8_t priority
```

This priority for AHB Master Read which this AHB RX Buffer is assigned.

```
uint8_t masterIndex
```

AHB Master ID the AHB RX Buffer is assigned.

```
uint16_t bufferSize
```

AHB buffer size in byte.

```
bool enablePrefetch
```

AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows prefetch disable/enable separately for each master.

```
struct _flexspi_ahbBuffers_ctrl
```

```
#include <fsl_flexspi.h> Structure to control all AHB buffers.
```

Public Members

```
flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]
```

Configurations of all AHB buffers.

```
struct _flexspi_config
```

```
#include <fsl_flexspi.h> FLEXSPI configuration structure.
```

Public Members

```
flexspi_read_sample_clock_t rxSampleClock
```

Sample Clock source selection for Flash Reading.

```
bool enableSckFreeRunning
```

Enable/disable SCK output free-running.

```
bool enableDoze
```

Enable/disable doze mode support.

```
bool enableHalfSpeedAccess
```

Enable/disable divide by 2 of the clock for half speed commands.

bool enableSameConfigForAll

Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16_t seqTimeoutCycle

Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCycle*1024 serial root clock cycles.

uint8_t ipGrantTimeoutCycle

Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle*1024 AHB clock cycles.

uint8_t txWatermark

FLEXSPI IP transmit watermark value.

uint8_t rxWatermark

FLEXSPI receive watermark value.

struct __flexspi_device_config

#include <fsl_flexspi.h> External device configuration items.

Public Members

uint32_t flexspiRootClk

FLEXSPI serial root clock.

bool isSck2Enabled

FLEXSPI use SCK2.

uint32_t flashSize

Flash size in KByte.

bool addressShift

Address shift.

flexspi_cs_interval_cycle_unit_t CSIntervalUnit

CS interval unit, 1 or 256 cycle.

uint16_t CSInterval

CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

uint8_t CSHoldTime

CS line hold time.

uint8_t CSSetupTime

CS line setup time.

uint8_t dataValidTime

Data valid time for external device.

uint8_t columnSpace

Column space size.

bool enableWordAddress

If enable word address.

uint8_t AWRSeqIndex

Sequence ID for AHB write command.

uint8_t AWRSeqNumber

Sequence number for AHB write command.

uint8_t ARDSeqIndex

Sequence ID for AHB read command.

uint8_t ARDSeqNumber

Sequence number for AHB read command.

flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit

AHB write wait unit.

uint16_t AHBWriteWaitInterval

AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask

Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

struct *_flexspi_transfer*

#include <fsl_flexspi.h> Transfer structure for FLEXSPI.

Public Members

uint32_t deviceAddress

Operation device address.

flexspi_port_t port

Operation port.

flexspi_command_type_t cmdType

Execution command type.

uint8_t seqIndex

Sequence ID for command.

uint8_t SeqNumber

Sequence number for command.

uint32_t *data

Data buffer.

size_t dataSize

Data size in bytes.

struct *_flexspi_handle*

#include <fsl_flexspi.h> Transfer handle structure for FLEXSPI.

Public Members

uint32_t state

Internal state for FLEXSPI transfer

uint8_t *data

Data buffer.

size_t dataSize

Remaining Data size in bytes.

size_t transferTotalSize

Total Data size in bytes.

flexspi_transfer_callback_t completionCallback

Callback for users while transfer finish or error occurred

void *userData

FLEXSPI callback function parameter.

struct *_flexspi_addr_map_config*

#include <fsl_flexspi.h> Address mapping configuration structure.

Public Members

uint32_t addrStart

Remapping start address.

uint32_t addrEnd

Remapping end address.

uint32_t addrOffset

Address offset.

bool remapEnable

Enable address remapping.

struct *ahbConfig*

Public Members

uint8_t ahbGrantTimeoutCycle

Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCycle*1024 AHB clock cycles.

uint16_t ahbBusTimeoutCycle

Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle*1024 AHB clock cycles.

uint8_t resumeWaitCycle

Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]

AHB buffer size.

bool enableClearAHBBufferOpt

Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt

Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

2.25 FLEXSPI DMA Driver

```
void FLEXSPI_TransferCreateHandleDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle,
                                     flexspi_dma_callback_t callback, void *userData,
                                     dma_handle_t *txDmaHandle, dma_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_dma_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_TransferUpdateSizeDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle,
                                   flexspi_dma_transfer_nsize_t nsize)
```

Update FLEXSPI DMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

See also:

flexspi_dma_transfer_nsize_t.

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_dma_handle_t structure
- nsize – FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_DMAnSize1Bytes(one byte).

```
status_t FLEXSPI_TransferDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle,
                             flexspi_transfer_t *xfer)
```

Transfers FLEXSPI data using an dma 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_dma_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 DMA transfer.
- kStatus_Success – FLEXSPI successfully start dma transfer.

`void FLEXSPI__TransferAbortDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle)`

Aborts the transfer data using dma.

This function aborts the transfer data using dma.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – Pointer to `flexspi_dma_handle_t` structure

`status_t FLEXSPI__TransferGetTransferCountDMA(FLEXSPI_Type *base, flexspi_dma_handle_t *handle, size_t *count)`

Gets the transferred counts of transfer.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – Pointer to `flexspi_dma_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_DMA_DRIVER_VERSION`

FLEXSPI DMA driver version 2.2.1.

`enum _flexspi_dma_ntransfer_size`

dma transfer configuration

Values:

enumerator `kFLEXPSI_DMAnSize1Bytes`

Source/Destination data transfer size is 1 byte every time

enumerator `kFLEXPSI_DMAnSize2Bytes`

Source/Destination data transfer size is 2 bytes every time

enumerator `kFLEXPSI_DMAnSize4Bytes`

Source/Destination data transfer size is 4 bytes every time

`typedef struct _flexspi_dma_handle flexspi_dma_handle_t`

`typedef void (*flexspi_dma_callback_t)(FLEXSPI_Type *base, flexspi_dma_handle_t *handle, status_t status, void *userData)`

FLEXSPI dma transfer callback function for finish and error.

`typedef enum _flexspi_dma_ntransfer_size flexspi_dma_transfer_nsize_t`

dma transfer configuration

`struct _flexspi_dma_handle`

`#include <fsl_flexspi_dma.h>` FLEXSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

`dma_handle_t *txDmaHandle`

dma handler for FLEXSPI Tx.

dma_handle_t *rxDmaHandle
dma handler for FLEXSPI Rx.

size_t transferSize
Bytes need to transfer.

flexspi_dma_transfer_nsize_t nsize
dma SSIZE/DSIZE in each transfer.

uint8_t nbytes
dma minor byte transfer count initially configured.

uint8_t count
The transfer data count in a DMA request.

uint32_t state
Internal state for FLEXSPI dma transfer.

flexspi_dma_callback_t completionCallback
A callback function called after the dma transfer is finished.

void *userData
User callback parameter

2.26 Flexspi_nor_flash_driver

Flash Pad Definitions.

Values:

enumerator kSerialFlash_1Pad

enumerator kSerialFlash_2Pads

enumerator kSerialFlash_4Pads

enumerator kSerialFlash_8Pads

FLEXSPI clock configuration type.

Values:

enumerator kFLEXSPIClk_SDR

Clock configure for SDR mode

enumerator kFLEXSPIClk_DDR

Clock configurat for DDR mode

enum _flexspi_read_sample_clk

FLEXSPI Read Sample Clock Source definition.

Values:

enumerator kFLEXSPIReadSampleClk_LoopbackInternally

enumerator kFLEXSPIReadSampleClk_LoopbackFromDqsPad

enumerator kFLEXSPIReadSampleClk_LoopbackFromSckPad

enumerator kFLEXSPIReadSampleClk_ExternalInputFromDqsPad

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

enumerator kFLEXSPISerialClk_30MHz

enumerator kFLEXSPISerialClk_50MHz

enumerator kFLEXSPISerialClk_60MHz

enumerator kFLEXSPISerialClk_75MHz

enumerator kFLEXSPISerialClk_80MHz

enumerator kFLEXSPISerialClk_100MHz

enumerator kFLEXSPISerialClk_133MHz

enumerator kFLEXSPISerialClk_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

enum _flexspi_status_groups

FLEXSPI status group numbers.

Values:

enumerator kStatusROMGroup_FLEXSPI

Group number for ROM FLEXSPI status codes.

enumerator kStatusROMGroup_FLEXSPINOR

ROM FLEXSPI NOR status group number.

enum _flexspi_nor_status

FLEXSPI NOR status.

Values:

enumerator kStatus_FLEXSPINOR_ProgramFail

Status for Page programming failure

enumerator kStatus_FLEXSPINOR_EraseSectorFail

Status for Sector Erase failure

enumerator kStatus_FLEXSPINOR_EraseAllFail

Status for Chip Erase failure

enumerator kStatus_FLEXSPINOR_WaitTimeout

Status for timeout

enumerator kStatus_FlexSPINOR_NotSupported

enumerator kStatus_FlexSPINOR_WriteAlignmentError

Status for Alignment error

enumerator kStatus_FlexSPINOR_CommandFailure

Status for Erase/Program Verify Error

enumerator kStatus_FlexSPINOR_SFDP_NotFound

Status for SFDP read failure

enumerator kStatus_FLEXSPINOR_Unsupported_SFDP_Version

Status for Unrecognized SFDP version

enumerator kStatus_FLEXSPINOR_Flash_NotFound

Status for Flash detection failure

enumerator kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed

Status for DDR Read dummy probe failure

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Status for Sequence Execution timeout

enumerator kStatus_FLEXSPI_InvalidSequence

Status for Invalid Sequence

enumerator kStatus_FLEXSPI_DeviceTimeout

Status for Device timeout

Configure the device_type of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorCfgOption_Tag

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_SDR

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_DDR

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH1V8

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH3V0

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalSDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalSDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalDDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalSDR

Configure the quad_mode_setting of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorQuadMode_NotConfig

enumerator kSerialNorQuadMode_StatusReg1_Bit6

enumerator kSerialNorQuadMode_StatusReg2_Bit1

enumerator kSerialNorQuadMode_StatusReg2_Bit7

enumerator kSerialNorQuadMode_StatusReg2_Bit1_0x31

FLEXSPI NOR Octal mode.

Values:

enumerator kSerialNorOctalMode_NoOctalEnableBit

enumerator kSerialNorOctalMode_HasOctalEnableBit

miscellaneous mode

Values:

enumerator kSerialNorEnhanceMode__Disabled

enumerator kSerialNorEnhanceMode__0_4_4_Mode

enumerator kSerialNorEnhanceMode__0_8_8_Mode

enumerator kSerialNorEnhanceMode__DataOrderSwapped

enumerator kSerialNorEnhanceMode__2ndPinMux

enumerator kSerialNorEnhanceMode__InternalLoopback

enumerator kSerialNorEnhanceMode__SpiMode

enumerator kSerialNorEnhanceMode__ExtDqs

FLEXSPI NOR reset logic options.

Values:

enumerator kFlashResetLogic__Disabled

enumerator kFlashResetLogic__ResetPin

enumerator kFlashResetLogic__JedecHwReset

Configure the flash_connection of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorConnection__SinglePortA

enumerator kSerialNorConnection__Parallel

enumerator kSerialNorConnection__SinglePortB

enumerator kSerialNorConnection__BothPorts

FLEXSPI ROOT clock source related definitions.

Values:

enumerator kFLEXSPIClkSrc__MainClk

enumerator kFLEXSPIClkSrc__Pll0

enumerator kFLEXSPIClkSrc__FroHf

enumerator kFLEXSPIClkSrc__Pll1

Restore sequence options Configure the restore_sequence of “flash_run_context_t” structure.

Values:

enumerator kRestoreSequence__None

enumerator kRestoreSequence__HW__Reset
enumerator kRestoreSequence__QPI_4_0xFFs
enumerator kRestoreSequence__QPI_Mode_0x00
enumerator kRestoreSequence__8QPI_FF
enumerator kRestoreSequence__Send_F0
enumerator kRestoreSequence__Send_66_99
enumerator kRestoreSequence__Send_6699_9966
enumerator kRestoreSequence__Send_06_FF
 Adesto EcoXIP
enumerator kRestoreSequence__QPI_5_0xFFs
enumerator kRestoreSequence__Send_QPI_8_0xFFs
enumerator kRestoreSequence__Wakeup_0xAB
enumerator kRestoreSequence__Wakeup_0xAB_54

Port mode options.

Values:

enumerator kFlashInstMode_ExtendedSpi
enumerator kFlashInstMode_0_4_4_SDR
enumerator kFlashInstMode_0_4_4_DDR
enumerator kFlashInstMode_DPI_SDR
enumerator kFlashInstMode_DPI_DDR
enumerator kFlashInstMode_QPI_SDR
enumerator kFlashInstMode_QPI_DDR
enumerator kFlashInstMode_OPI_SDR
enumerator kFlashInstMode_OPI_DDR

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_operation*

flexspi operation.

Values:

enumerator *kFLEXSPIOperation_Command*

Only command, both TX and RX buffer are ignored.

enumerator *kFLEXSPIOperation_Config*

Configure device mode, the TX FIFO size is fixed in LUT.

enumerator *kFLEXSPIOperation_Write*

Write, only TX buffer is effective

enumerator *kFLEXSPIOperation_Read*

Read, only Rx Buffer is effective.

enumerator *kFLEXSPIOperation_End*

enum *flexspi_clock_type_t*

FLEXSPI Clock Type.

Values:

enumerator *kFlexSpiClock_CoreClock*

ARM Core Clock

enumerator *kFlexSpiClock_AhbClock*

AHB clock

enumerator *kFlexSpiClock_SerialRootClock*

Serial Root Clock

enumerator *kFlexSpiClock_IpgClock*

IPG clock

typedef struct *_serial_nor_config_option* *serial_nor_config_option_t*

Serial NOR configuration option.

typedef struct *_lut_sequence* *flexspi_lut_seq_t*

FLEXSPI LUT Sequence structure.

typedef struct *_FlexSPIConfig* *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.

typedef struct *_flexspi_xfer* *flexspi_xfer_t*

FLEXSPI Transfer Context.

status_t 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 succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t FLEXSPI_NorFlash_ProgramPage(uint32_t instance, flexspi_nor_config_t *config, uint32_t dstAddr, 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.

Parameters

- `instance` – storage the instance of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `dstAddr` – A pointer to the desired flash memory to be programmed. 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.
- `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 succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t 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 succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t FLEXSPI_NorFlash_EraseSector(uint32_t instance, flexspi_nor_config_t *config, uint32_t address)`

Erase one sector specified by address.

This function erases one of NOR flash sectors based on the desired address.

Parameters

- `instance` – storage the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `address` – The start address of the desired NOR flash memory to be erased.
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.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t FLEXSPI_NorFlash_EraseBlock(uint32_t instance, flexspi_nor_config_t *config, uint32_t address)`

Erase one block specified by address.

This function erases one block of NOR flash based on the desired address.

Parameters

- `instance` – storage the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `address` – The start address of the desired NOR flash memory to be erased.
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.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t 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 succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.

- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t 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.

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. 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.
- `length` – The length, given in bytes to be erased. 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.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t FLEXSPI_NorFlash_Read(uint32_t instance, flexspi_nor_config_t *config, uint32_t *dst, uint32_t start, uint32_t bytes)`

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.

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. 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.
- `start` – The start address of the desired NOR flash memory to be read.
- `bytes` – The length, given in bytes to be read.

Return values

- `kStatus_Success` – Api was executed successfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.

- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t 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 succesfully.
- `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 FLEXSPI_NorFlash_UpdateLut(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t numberOfSeq)`

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.
- `numberOfSeq` – storage sequence number.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `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 FLEXSPI_NorFlash_SetClockSource(uint32_t clockSource)`

Set the clock source for FLEXSPI NOR.

Parameters

- `clockSource` – Clock source for FLEXSPI NOR. See to “`flexspi_nor_clock_source`”.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.

`void FLEXSPI_NorFlash_ConfigClock(uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode)`

config flexspi clock

Parameters

- instance – storage the index of FLEXSPI.
- freqOption – pointer to FlexSPIFlexSPI flash serial clock frequency.
- sampleClkMode – pointer to configure the FlexSPI clock configuration type.

FLEXSPI_FEATURE_HAS_PARALLEL_MODE

FLEXSPI Feature related definitions

FSL_ROM_FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

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

FLEXSPI_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: The 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 blobk

struct `_serial_nor_config_option`

`#include <fsl_flexspi_nor_flash.h>` Serial NOR configuration option.

union `flash_run_context_t`

`#include <fsl_flexspi_nor_flash.h>`

Public Members

struct `flash_run_context_t` B

uint32_t U

struct `_lut_sequence`

`#include <fsl_flexspi_nor_flash.h>` FLEXSPI LUT Sequence structure.

Public Members

uint8_t seqNum
Sequence Number, valid number: 1-16

uint8_t seqId
Sequence Index, valid number: 0-15

struct flexspi_dll_time_t
#include <fsl_flexspi_nor_flash.h>

Public Members

uint8_t time_100ps
Data valid time, in terms of 100ps

uint8_t delay_cells
Data valid time, in terms of delay cells

struct __FlexSPIConfig
#include <fsl_flexspi_nor_flash.h> FLEXSPI Memory Configuration Block.

Public Members

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] CS hold time, default value: 3

uint8_t csSetupTime
[0x00e-0x00e] CS 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 device-ModeSeq

`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 dll1CrVal`
 [0x1b0-0x1b3] Customizable DLL0CR setting */

`uint32_t reserved4[2]`
 [0x1b4-0x1b7] Customizable DLL1CR setting */
 [0x1b8-0x1bf] Reserved for future use

`struct _flexspi_nor_config`
 #include <fsl_flexspi_nor_flash.h> Serial NOR configuration block.

Public Members

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 commmand execution

uint32_t blockSize

Block size

uint32_t flashStateCtx

Flash State Context

uint32_t reserve2[10]

Reserved for future use

struct __flexspi_xfer

#include <fsl_flexspi_nor_flash.h> FLEXSPI Transfer Context.

Public Members

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

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

struct B

2.27 FREQME: Frequency Measurement

2.28 GINT: Group GPIO Input Interrupt Driver

FSL_GINT_DRIVER_VERSION

Driver version.

enum __gint_comb

GINT combine inputs type.

Values:

enumerator kGINT_CombineOr

A grouped interrupt is generated when any one of the enabled inputs is active

enumerator kGINT_CombineAnd

A grouped interrupt is generated when all enabled inputs are active

enum __gint_trig

GINT trigger type.

Values:

enumerator kGINT_TrigEdge

Edge triggered based on polarity

enumerator kGINT_TrigLevel

Level triggered based on polarity

enum __gint_port

Values:

enumerator kGINT_Port0

typedef enum __gint_comb gint_comb_t

GINT combine inputs type.

typedef enum __gint_trig gint_trig_t

GINT trigger type.

typedef enum __gint_port gint_port_t

typedef void (*gint_cb_t)(void)

GINT Callback function.

void GINT_Init(GINT_Type *base)

Initialize GINT peripheral.

This function initializes the GINT peripheral and enables the clock.

Parameters

- base – Base address of the GINT peripheral.

Return values

None. –

void GINT_SetCtrl(GINT_Type *base, gint_comb_t comb, gint_trig_t trig, gint_cb_t callback)

Setup GINT peripheral control parameters.

This function sets the control parameters of GINT peripheral.

Parameters

- base – Base address of the GINT peripheral.

- `comb` – Controls if the enabled inputs are logically ORed or ANDed for interrupt generation.
- `trig` – Controls if the enabled inputs are level or edge sensitive based on polarity.
- `callback` – This function is called when configured group interrupt is generated.

Return values

None. –

`void GINT_GetCtrl(GINT_Type *base, gint_comb_t *comb, gint_trig_t *trig, gint_cb_t *callback)`
Get GINT peripheral control parameters.

This function returns the control parameters of GINT peripheral.

Parameters

- `base` – Base address of the GINT peripheral.
- `comb` – Pointer to store combine input value.
- `trig` – Pointer to store trigger value.
- `callback` – Pointer to store callback function.

Return values

None. –

`void GINT_ConfigPins(GINT_Type *base, gint_port_t port, uint32_t polarityMask, uint32_t enableMask)`

Configure GINT peripheral pins.

This function enables and controls the polarity of enabled pin(s) of a given port.

Parameters

- `base` – Base address of the GINT peripheral.
- `port` – Port number.
- `polarityMask` – Each bit position selects the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.
- `enableMask` – Each bit position selects if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

Return values

None. –

`void GINT_GetConfigPins(GINT_Type *base, gint_port_t port, uint32_t *polarityMask, uint32_t *enableMask)`

Get GINT peripheral pin configuration.

This function returns the pin configuration of a given port.

Parameters

- `base` – Base address of the GINT peripheral.
- `port` – Port number.
- `polarityMask` – Pointer to store the polarity mask Each bit position indicates the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH.
- `enableMask` – Pointer to store the enable mask. Each bit position indicates if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.

Return values

None. –

void GINT_EnableCallback(GINT_Type *base)

Enable callback.

This function enables the interrupt for the selected GINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

- base – Base address of the GINT peripheral.

Return values

None. –

void GINT_DisableCallback(GINT_Type *base)

Disable callback.

This function disables the interrupt for the selected GINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

- base – Base address of the peripheral.

Return values

None. –

static inline void GINT_ClrStatus(GINT_Type *base)

Clear GINT status.

This function clears the GINT status bit.

Parameters

- base – Base address of the GINT peripheral.

Return values

None. –

static inline uint32_t GINT_GetStatus(GINT_Type *base)

Get GINT status.

This function returns the GINT status.

Parameters

- base – Base address of the GINT peripheral.

Return values

status – = 0 No group interrupt request. = 1 Group interrupt request active.

void GINT_Deinit(GINT_Type *base)

Deinitialize GINT peripheral.

This function disables the GINT clock.

Parameters

- base – Base address of the GINT peripheral.

Return values

None. –

2.29 Hashcrypt: The Cryptographic Accelerator

2.30 Hashcrypt Background HASH

```
void HASHCRYPT_SHA_SetCallback(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx,  
                              hashcrypt_callback_t callback, void *userData)
```

Initializes the HASHCRYPT handle for background hashing.

This function initializes the hash context for background hashing (Non-blocking) APIs. This is less typical interface to hash function, but can be used for parallel processing, when main CPU has something else to do. Example is digital signature RSASSA-PKCS1-V1_5-VERIFY((n,e),M,S) algorithm, where background hashing of M can be started, then CPU can compute $S^e \bmod n$ (in parallel with background hashing) and once the digest becomes available, CPU can proceed to comparison of EM with EM'.

Parameters

- base – HASHCRYPT peripheral base address.
- ctx – **[out]** Hash context.
- callback – Callback function.
- userData – User data (to be passed as an argument to callback function, once callback is invoked from isr).

```
status_t HASHCRYPT_SHA_UpdateNonBlocking(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t  
                                         *ctx, const uint8_t *input, size_t inputSize)
```

Create running hash on given data.

Configures the HASHCRYPT to compute new running hash as AHB master and returns immediately. HASHCRYPT AHB Master mode supports only aligned input address and can be called only once per continuous block of data. Every call to this function must be preceded with HASHCRYPT_SHA_Init() and finished with HASHCRYPT_SHA_Finish(). Once callback function is invoked by HASHCRYPT isr, it should set a flag for the main application to finalize the hashing (padding) and to read out the final digest by calling HASHCRYPT_SHA_Finish().

Parameters

- base – HASHCRYPT peripheral base address
- ctx – Specifies callback. Last incomplete 512-bit block of the input is copied into clear buffer for padding.
- input – 32-bit word aligned pointer to Input data.
- inputSize – Size of input data in bytes (must be word aligned)

Returns

Status of the hash update operation.

2.31 Hashcrypt common functions

FSL_HASHCRYPT_DRIVER_VERSION

HASHCRYPT driver version. Version 2.2.16.

Current version: 2.2.16

Change log:

- Version 2.0.0

- Initial version
- Version 2.0.1
 - Support loading AES key from unaligned address
- Version 2.0.2
 - Support loading AES key from unaligned address for different compiler and core variants
- Version 2.0.3
 - Remove SHA512 and AES ICB algorithm definitions
- Version 2.0.4
 - Add SHA context switch support
- Version 2.1.0
 - Update the register name and macro to align with new header.
- Version 2.1.1
 - Fix MISRA C-2012.
- Version 2.1.2
 - Support loading AES input data from unaligned address.
- Version 2.1.3
 - Fix MISRA C-2012.
- Version 2.1.4
 - Fix context switch cannot work when switching from AES.
- Version 2.1.5
 - Add data synchronization barrier inside `hashcrypt_sha_ldm_stm_16_words()` to prevent possible optimization issue.
- Version 2.2.0
 - Add AES-OFB and AES-CFB mixed IP/SW modes.
- Version 2.2.1
 - Add data synchronization barrier inside `hashcrypt_sha_ldm_stm_16_words()` prevent compiler from reordering memory write when `-O2` or higher is used.
- Version 2.2.2
 - Add data synchronization barrier inside `hashcrypt_sha_ldm_stm_16_words()` to fix optimization issue
- Version 2.2.3
 - Added check for size in `hashcrypt_aes_one_block` to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.
- Version 2.2.4
 - In all `HASHCRYPT_AES_xx` functions have been added setting CTRL_MODE bitfield to 0 after processing data, which decreases power consumption.
- Version 2.2.5
 - Add data synchronization barrier and instruction synchronization barrier inside `hashcrypt_sha_process_message_data()` to fix optimization issue
- Version 2.2.6

- Add data synchronization barrier inside `HASHCRYPT_SHA_Update()` and `hashcrypt_get_data()` function to fix optimization issue on MDK and ARMGCC release targets
- Version 2.2.7
 - Add data synchronization barrier inside `HASHCRYPT_SHA_Update()` to fix optimization issue on MCUX IDE release target
- Version 2.2.8
 - Unify hashcrypt hashing behavior between aligned and unaligned input data
- Version 2.2.9
 - Add handling of set ERROR bit in the STATUS register
- Version 2.2.10
 - Fix missing error statement in `hashcrypt_save_running_hash()`
- Version 2.2.11
 - Fix incorrect SHA-256 calculation for long messages with reload
- Version 2.2.12
 - Fix hardfault issue on the Keil compiler due to unaligned `memcpy()` input on some optimization levels
- Version 2.2.13
 - Added function `hashcrypt_seed_prng()` which loading random number into PRNG_SEED register before AES operation for SCA protection
- Version 2.2.14
 - Modify function `hashcrypt_get_data()` to prevent issue with unaligned access
- Version 2.2.15
 - Add wait on DIGEST BIT inside `hashcrypt_sha_one_block()` to fix issues with some optimization flags
- Version 2.2.16
 - Add DSB instruction inside `hashcrypt_sha_ldm_stm_16_words()` to fix issues with some optimization flags
- Version 2.2.17
 - Fix context size when hashcrypt built with reload feature

`enum _hashcrypt_algo_t`

Algorithm used for Hashcrypt operation.

Values:

enumerator `kHASHCRYPT_Sha1`
SHA_1

enumerator `kHASHCRYPT_Sha256`
SHA_256

enumerator `kHASHCRYPT_Aes`
AES

`typedef enum _hashcrypt_algo_t hashcrypt_algo_t`

Algorithm used for Hashcrypt operation.

`void HASHCRYPT__Init(HASHCRYPT_Type *base)`

Enables clock and disables reset for HASHCRYPT peripheral.

Enable clock and disable reset for HASHCRYPT.

Parameters

- `base` – HASHCRYPT base address

`void HASHCRYPT__Deinit(HASHCRYPT_Type *base)`

Disables clock for HASHCRYPT peripheral.

Disable clock and enable reset.

Parameters

- `base` – HASHCRYPT base address

`HASHCRYPT__MODE_SHA1`

Algorithm definitions correspond with the values for Mode field in Control register !

`HASHCRYPT__MODE_SHA256`

`HASHCRYPT__MODE_AES`

2.32 Hashcrypt AES

`enum _hashcrypt_aes_mode_t`

AES mode.

Values:

enumerator `kHASHCRYPT__AesEcb`

AES ECB mode

enumerator `kHASHCRYPT__AesCbc`

AES CBC mode

enumerator `kHASHCRYPT__AesCtr`

AES CTR mode

`enum _hashcrypt_aes_keysize_t`

Size of AES key.

Values:

enumerator `kHASHCRYPT__Aes128`

AES 128 bit key

enumerator `kHASHCRYPT__Aes192`

AES 192 bit key

enumerator `kHASHCRYPT__Aes256`

AES 256 bit key

enumerator `kHASHCRYPT__InvalidKey`

AES invalid key

`enum _hashcrypt_key`

HASHCRYPT key source selection.

Values:

enumerator kHASHCRYPT_UserKey

HASHCRYPT user key

enumerator kHASHCRYPT_SecretKey

HASHCRYPT secret key (dedicated hw bus from PUF)

typedef enum *_hashcrypt_aes_mode_t* hashcrypt_aes_mode_t
AES mode.

typedef enum *_hashcrypt_aes_keysize_t* hashcrypt_aes_keysize_t
Size of AES key.

typedef enum *_hashcrypt_key* hashcrypt_key_t
HASHCRYPT key source selection.

typedef struct *_hashcrypt_handle* hashcrypt_handle_t

struct *_hashcrypt_handle* ____attribute__((aligned))

status_t HASHCRYPT_AES_SetKey(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle,
const uint8_t *key, size_t keySize)

Set AES key to *hashcrypt_handle_t* struct and optionally to HASHCRYPT.

Sets the AES key for encryption/decryption with the *hashcrypt_handle_t* structure. The *hashcrypt_handle_t* input argument specifies key source.

Parameters

- base – HASHCRYPT peripheral base address.
- handle – Handle used for the request.
- key – 0-mod-4 aligned pointer to AES key.
- keySize – AES key size in bytes. Shall equal 16, 24 or 32.

Returns

status from set key operation

status_t HASHCRYPT_AES_EncryptEcb(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle,
const uint8_t *plaintext, uint8_t *ciphertext, size_t
size)

Encrypts AES on one or multiple 128-bit block(s).

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- 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.

Returns

Status from encrypt operation

status_t HASHCRYPT_AES_DecryptEcb(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle,
const uint8_t *ciphertext, uint8_t *plaintext, size_t
size)

Decrypts AES on one or multiple 128-bit block(s).

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- ciphertext – Input plain text to encrypt
- plaintext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from decrypt operation

```
status_t HASHCRYPT_AES_EncryptCbc(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,
                                   const uint8_t *plaintext, uint8_t *ciphertext, size_t
                                   size, const uint8_t iv[16])
```

Encrypts AES using CBC block mode.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- 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.

Returns

Status from encrypt operation

```
status_t HASHCRYPT_AES_DecryptCbc(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,
                                   const uint8_t *ciphertext, uint8_t *plaintext, size_t
                                   size, const uint8_t iv[16])
```

Decrypts AES using CBC block mode.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- 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.

Returns

Status from decrypt operation

```
status_t HASHCRYPT_AES_CryptCtr(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,
                                 const uint8_t *input, uint8_t *output, size_t size, uint8_t
                                 counter[16U], uint8_t counterlast[16U], 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* – HASHCRYPT peripheral base address
- *handle* – Handle used for this request.
- *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)
- *counterlast* – **[out]** Output cipher of last counter, for chained CTR calls (statefull encryption). 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 HASHCRYPT_AES_CryptOfb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                const uint8_t *input, uint8_t *output, size_t size, const  
                                uint8_t iv[16U])
```

Encrypts or decrypts AES using OFB block mode.

Encrypts or decrypts AES using OFB block mode. AES OFB 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* – HASHCRYPT peripheral base address
- *handle* – Handle used for this request.
- *input* – Input data for OFB block mode
- *output* – **[out]** Output data for OFB block mode
- *size* – Size of input and output data in bytes
- *iv* – Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

```
status_t HASHCRYPT_AES_EncryptCfb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,  
                                  const uint8_t *plaintext, uint8_t *ciphertext, size_t size,  
                                  const uint8_t iv[16])
```

Encrypts AES using CFB block mode.

Parameters

- *base* – HASHCRYPT peripheral base address
- *handle* – Handle used for this request.
- *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.

Returns

Status from encrypt operation

```
status_t HASHCRYPT_AES_DecryptCfb(HASHCRYPT_Type *base, hashcrypt_handle_t *handle,
                                  const uint8_t *ciphertext, uint8_t *plaintext, size_t size,
                                  const uint8_t iv[16])
```

Decrypts AES using CFB block mode.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plaintext 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.

Returns

Status from encrypt operation

```
HASHCRYPT_AES_BLOCK_SIZE
```

AES block size in bytes

```
AES_ENCRYPT
```

```
AES_DECRYPT
```

```
struct __hashcrypt_handle
```

#include <fsl_hashcrypt.h> Specify HASHCRYPT's key resource.

Public Members

```
uint32_t keyWord[8]
```

Copy of user key (set by HASHCRYPT_AES_SetKey()).

```
hashcrypt_key_t keyType
```

For operations with key (such as AES encryption/decryption), specify key type.

2.33 Hashcrypt HASH

```
typedef struct __hashcrypt_hash_ctx_t hashcrypt_hash_ctx_t
```

Storage type used to save hash context.

```
typedef void (*hashcrypt_callback_t)(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx,
status_t status, void *userData)
```

HASHCRYPT background hash callback function.

```
status_t HASHCRYPT_SHA(HASHCRYPT_Type *base, hashcrypt_algo_t algo, const uint8_t
*input, size_t inputSize, uint8_t *output, size_t outputSize)
```

Create HASH on given data.

Perform the full SHA in one function call. The function is blocking.

Parameters

- base – HASHCRYPT peripheral base address
- algo – Underlying algorithm to use for hash computation.
- input – Input data
- inputSize – Size of input data 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.

```
status_t HASHCRYPT_SHA_Init(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx,  
                           hashcrypt_algo_t algo)
```

Initialize HASH context.

This function initializes the HASH.

Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[out]** Output hash context
- algo – Underlying algorithm to use for hash computation.

Returns

Status of initialization

```
status_t HASHCRYPT_SHA_Update(HASHCRYPT_Type *base, hashcrypt_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 has been updated (HASHCRYPT has processed the input data), so the memory at *input* pointer can be released back to system. The HASHCRYPT context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[inout]** HASH context
- input – Input data
- inputSize – Size of input data in bytes

Returns

Status of the hash update operation

```
status_t HASHCRYPT_SHA_Finish(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, uint8_t  
                             *output, size_t *outputSize)
```

Finalize hashing.

Outputs the final hash (computed by HASHCRYPT_HASH_Update()) and erases the context.

Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[inout]** Input hash context
- output – **[out]** Output hash data

- `outputSize` – **[inout]** Optional parameter (can be passed as NULL). On function entry, it specifies the size of `output[]` buffer. On function return, it stores the number of updated output bytes.

Returns

Status of the hash finish operation

`HASHCRYPT_HASH_CTX_SIZE`

HASHCRYPT HASH Context size.

`struct __hashcrypt_hash_ctx_t`

`#include <fsl_hashcrypt.h>` Storage type used to save hash context.

Public Members

`uint32_t x[22]`

storage

2.34 HSCMP: High Speed Analog Comparator Driver

`void HSCMP_Init(HSCMP_Type *base, const hscmp_config_t *config)`

Initialize the HSCMP.

This function initializes the HSCMP module. The operations included are:

- Enabling the clock for HSCMP module.
- Configuring the comparator.
- Enabling the HSCMP module. Note: For some devices, multiple HSCMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the HSCMPs. Check the chip reference manual for the clock assignment of the HSCMP.

Parameters

- `base` – HSCMP peripheral base address.
- `config` – Pointer to “`hscmp_config_t`” structure.

`void HSCMP_Deinit(HSCMP_Type *base)`

De-initializes the HSCMP module.

This function de-initializes the HSCMP module. The operations included are:

- Disabling the HSCMP module.
- Disabling the clock for HSCMP module.

This function disables the clock for the HSCMP. Note: For some devices, multiple HSCMP instance shares the same clock gate. In this case, before disabling the clock for the HSCMP, ensure that all the HSCMP instances are not used.

Parameters

- `base` – HSCMP peripheral base address.

`void HSCMP_GetDefaultConfig(hscmp_config_t *config)`

Gets an available pre-defined settings for the comparator’s configuration.

This function initializes the comparator configuration structure to these default values:

```
config->enableStopMode    = false;
config->enableOutputPin    = false;
config->useUnfilteredOutput = false;
config->enableInvertOutput = false;
config->hysteresisMode     = kHSCMP_HysteresisLevel0;
config->powerMode          = kHSCMP_LowSpeedPowerMode;
```

Parameters

- config – Pointer to “hscmp_config_t” structure.

static inline void HSCMP_Enable(HSCMP_Type *base, bool enable)

Enable/Disable HSCMP module.

Parameters

- base – HSCMP peripheral base address.
- enable – “true” means enable the module, and “false” means disable the module.

void HSCMP_SetInputChannels(HSCMP_Type *base, uint32_t positiveChannel, uint32_t negativeChannel)

Select the input channels for HSCMP. This function determines which input is selected for the negative and positive mux.

Parameters

- base – HSCMP peripheral base address.
- positiveChannel – Positive side input channel number. Available range is 0-7.
- negativeChannel – Negative side input channel number. Available range is 0-7.

static inline void HSCMP_EnableDMA(HSCMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events. Normally, the HSCMP generates a CPU interrupt if there is a rising/falling event. When DMA support is enabled and the rising/falling interrupt is enabled, the rising/falling event forces a DMA transfer request rather than a CPU interrupt instead.

Parameters

- base – HSCMP peripheral base address.
- enable – “true” means enable DMA support, and “false” means disable DMA support.

static inline void HSCMP_EnableWindowMode(HSCMP_Type *base, bool enable)

Enable/Disable window mode. When any windowed mode is active, COUTA is clocked by the bus clock whenever WINDOW = 1. The last latched value is held when WINDOW = 0. The optionally inverted comparator output COUT_RAW is sampled on every bus clock when WINDOW=1 to generate COUTA.

Parameters

- base – HSCMP peripheral base address.
- enable – “true” means enable window mode, and “false” means disable window mode.

void HSCMP_SetFilterConfig(HSCMP_Type *base, const *hscmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – HSCMP peripheral base address.
- config – Pointer to “hscmp_filter_config_t” structure.

void HSCMP_SetDACConfig(HSCMP_Type *base, const hscmp_dac_config_t *config)

Configure the internal DAC module.

Parameters

- base – HSCMP peripheral base address.
- config – Pointer to “hscmp_dac_config_t” structure. If config is “NULL”, disable internal DAC.

static inline void HSCMP_EnableInterrupts(HSCMP_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – HSCMP peripheral base address.
- mask – Mask value for interrupts. See “_hscmp_interrupt_enable”.

static inline void HSCMP_DisableInterrupts(HSCMP_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – HSCMP peripheral base address.
- mask – Mask value for interrupts. See “_hscmp_interrupt_enable”.

static inline uint32_t HSCMP_GetStatusFlags(HSCMP_Type *base)

Get the HSCMP status flags.

Parameters

- base – HSCMP peripheral base address.

Returns

Mask value for the asserted flags. See “_hscmp_status_flags”.

static inline void HSCMP_ClearStatusFlags(HSCMP_Type *base, uint32_t mask)

Clear the HSCMP status flags.

Parameters

- base – HSCMP peripheral base address.
- mask – Mask value for the flags. See “_hscmp_status_flags”.

FSL_HSCMP_DRIVER_VERSION

HSCMP driver version 2.0.4.

enum _hscmp_status_flags

HSCMP status flags mask.

Values:

enumerator kHSCMP_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kHSCMP_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kHSCMP_OutputAssertEventFlag

Return the current value of the analog comparator output. The flag does not support W1C.

enum `_hscmp_interrupt_enable`

HSCMP interrupt enable/disable mask.

Values:

enumerator `kHSCMP_OutputRisingInterruptEnable`

Comparator interrupt enable rising.

enumerator `kHSCMP_OutputFallingInterruptEnable`

Comparator interrupt enable falling.

enum `_hscmp_hysteresis_mode`

HSCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

Values:

enumerator `kHSCMP_HysteresisLevel0`

The hard block output has level 0 hysteresis internally.

enumerator `kHSCMP_HysteresisLevel1`

The hard block output has level 1 hysteresis internally.

enumerator `kHSCMP_HysteresisLevel2`

The hard block output has level 2 hysteresis internally.

enumerator `kHSCMP_HysteresisLevel3`

The hard block output has level 3 hysteresis internally.

enum `_hscmp_power_mode`

HSCMP nano mode.

Values:

enumerator `kHSCMP_LowSpeedPowerMode`

Low speed comparison mode is selected.

enumerator `kHSCMP_HighSpeedPowerMode`

High speed comparison mode is selected.

enumerator `kHSCMP_NanoPowerMode`

Nano power comparator is enabled.

enum `_hscmp_dac_reference_voltage_source`

Internal DAC reference voltage source.

Values:

enumerator `kHSCMP_VrefSourceVin1`

`vrefh_int` is selected as resistor ladder network supply reference `Vin`.

enumerator `kHSCMP_VrefSourceVin2`

`vrefh_ext` is selected as resistor ladder network supply reference `Vin`.

typedef enum `_hscmp_hysteresis_mode` `hscmp_hysteresis_mode_t`

HSCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

typedef enum `_hscmp_power_mode` `hscmp_power_mode_t`

HSCMP nano mode.

typedef enum `_hscmp_dac_reference_voltage_source` `hscmp_dac_reference_voltage_source_t`

Internal DAC reference voltage source.

```
typedef struct _hscmp_filter_config hscmp_filter_config_t
```

Configure the filter.

```
typedef struct _hscmp_dac_config hscmp_dac_config_t
```

configure the internal DAC.

```
typedef struct _hscmp_config hscmp_config_t
```

Configures the comparator.

```
struct _hscmp_filter_config
```

#include <fsl_hscmp.h> Configure the filter.

Public Members

bool enableSample

Decide whether to use the external SAMPLE as a sampling clock input.

uint8_t filterSampleCount

Filter Sample Count. Available range is 1-7; 0 disables the filter.

uint8_t filterSamplePeriod

Filter Sample Period. The divider to the bus clock. Available range is 0-255. The sampling clock must be at least 4 times slower than the system clock to the comparator. So if enableSample is “false”, filterSamplePeriod should be set greater than 4.

```
struct _hscmp_dac_config
```

#include <fsl_hscmp.h> configure the internal DAC.

Public Members

bool enableLowPowerMode

Decide whether to enable DAC low power mode.

hscmp_dac_reference_voltage_source_t referenceVoltageSource

Internal DAC supply voltage reference source.

uint8_t DACValue

Value for the DAC Output Voltage. Available range is 0-63.

```
struct _hscmp_config
```

#include <fsl_hscmp.h> Configures the comparator.

Public Members

bool enableStopMode

Decide whether to enable the comparator when in STOP modes.

bool enableOutputPin

Decide whether to enable the comparator is available in selected pin.

bool useUnfilteredOutput

Decide whether to use unfiltered output.

bool enableInvertOutput

Decide whether to inverts the comparator output.

hscmp_hysteresis_mode_t hysteresisMode

HSCMP hysteresis mode.

hscmp_power_mode_t powerMode
HSCMP power mode.

2.35 I2C: Inter-Integrated Circuit Driver

2.36 I2C DMA Driver

```
void I2C_MasterTransferCreateHandleDMA(I2C_Type *base, i2c_master_dma_handle_t *handle,  
                                       i2c_master_dma_transfer_callback_t callback, void  
                                       *userData, dma_handle_t *dmaHandle)
```

Init the I2C handle which is used in transactional functions.

Parameters

- base – I2C peripheral base address
- handle – pointer to *i2c_master_dma_handle_t* structure
- callback – pointer to user callback function
- userData – user param passed to the callback function
- dmaHandle – DMA handle pointer

```
status_t I2C_MasterTransferDMA(I2C_Type *base, i2c_master_dma_handle_t *handle,  
                               i2c_master_transfer_t *xfer)
```

Performs a master dma non-blocking transfer on the I2C bus.

Parameters

- base – I2C peripheral base address
- handle – pointer to *i2c_master_dma_handle_t* structure
- xfer – pointer to transfer structure of *i2c_master_transfer_t*

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStatus_I2C_Nak – Transfer error, receive Nak during transfer.

```
status_t I2C_MasterTransferGetCountDMA(I2C_Type *base, i2c_master_dma_handle_t *handle,  
                                       size_t *count)
```

Get master transfer status during a dma non-blocking transfer.

Parameters

- base – I2C peripheral base address
- handle – pointer to *i2c_master_dma_handle_t* structure
- count – Number of bytes transferred so far by the non-blocking transaction.

```
void I2C_MasterTransferAbortDMA(I2C_Type *base, i2c_master_dma_handle_t *handle)
```

Abort a master dma non-blocking transfer in a early time.

Parameters

- base – I2C peripheral base address

- handle – pointer to `i2c_master_dma_handle_t` structure

`FSL_I2C_DMA_DRIVER_VERSION`

I2C DMA driver version.

`typedef struct _i2c_master_dma_handle i2c_master_dma_handle_t`

I2C master dma handle typedef.

`typedef void (*i2c_master_dma_transfer_callback_t)(I2C_Type *base, i2c_master_dma_handle_t *handle, status_t status, void *userData)`

I2C master dma transfer callback typedef.

`typedef void (*flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type *base, i2c_master_dma_handle_t *handle)`

Typedef for master dma handler.

`I2C_MAX_DMA_TRANSFER_COUNT`

Maximum length of single DMA transfer (determined by capability of the DMA engine)

`struct _i2c_master_dma_handle`

`#include <fsl_i2c_dma.h>` I2C master dma transfer structure.

Public Members

`uint8_t state`

Transfer state machine current state.

`uint32_t transferCount`

Indicates progress of the transfer

`uint32_t remainingBytesDMA`

Remaining byte count to be transferred using DMA.

`uint8_t *buf`

Buffer pointer for current state.

`bool checkAddrNack`

Whether to check the nack signal is detected during addressing.

`dma_handle_t *dmaHandle`

The DMA handler used.

`i2c_master_transfer_t transfer`

Copy of the current transfer info.

`i2c_master_dma_transfer_callback_t completionCallback`

Callback function called after dma transfer finished.

`void *userData`

Callback parameter passed to callback function.

2.37 I2C Driver

`FSL_I2C_DRIVER_VERSION`

I2C driver version.

I2C status return codes.

Values:

enumerator kStatus_I2C_Busy

The master is already performing a transfer.

enumerator kStatus_I2C_Idle

The slave driver is idle.

enumerator kStatus_I2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_I2C_InvalidParameter

Unable to proceed due to invalid parameter.

enumerator kStatus_I2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_I2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_I2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_I2C_DmaRequestFail

DMA request failed.

enumerator kStatus_I2C_StartStopError

Start and stop error.

enumerator kStatus_I2C_UnexpectedState

Unexpected state.

enumerator kStatus_I2C_Timeout

Timeout when waiting for I2C master/slave pending status to set to continue transfer.

enumerator kStatus_I2C_Addr_Nak

NAK received for Address

enumerator kStatus_I2C_EventTimeout

Timeout waiting for bus event.

enumerator kStatus_I2C_SclLowTimeout

Timeout SCL signal remains low.

enum _i2c_status_flags

I2C status flags.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2C_MasterPendingFlag

The I2C module is waiting for software interaction. bit 0

enumerator kI2C_MasterArbitrationLostFlag

The arbitration of the bus was lost. There was collision on the bus. bit 4

enumerator kI2C_MasterStartStopErrorFlag

There was an error during start or stop phase of the transaction. bit 6

enumerator `ki2C_MasterIdleFlag`
The I2C master idle status. bit 5

enumerator `ki2C_MasterRxReadyFlag`
The I2C master rx ready status. bit 1

enumerator `ki2C_MasterTxReadyFlag`
The I2C master tx ready status. bit 2

enumerator `ki2C_MasterAddrNackFlag`
The I2C master address nack status. bit 7

enumerator `ki2C_MasterDataNackFlag`
The I2C master data nack status. bit 3

enumerator `ki2C_SlavePendingFlag`
The I2C module is waiting for software interaction. bit 8

enumerator `ki2C_SlaveNotStretching`
Indicates whether the slave is currently stretching clock (0 = yes, 1 = no). bit 11

enumerator `ki2C_SlaveSelected`
Indicates whether the slave is selected by an address match. bit 14

enumerator `ki2C_SaveDeselected`
Indicates that slave was previously deselected (deselect event took place, w1c). bit 15

enumerator `ki2C_SlaveAddressedFlag`
One of the I2C slave's 4 addresses is matched. bit 22

enumerator `ki2C_SlaveReceiveFlag`
Slave receive data available. bit 9

enumerator `ki2C_SlaveTransmitFlag`
Slave data can be transmitted. bit 10

enumerator `ki2C_SlaveAddress0MatchFlag`
Slave address0 match. bit 20

enumerator `ki2C_SlaveAddress1MatchFlag`
Slave address1 match. bit 12

enumerator `ki2C_SlaveAddress2MatchFlag`
Slave address2 match. bit 13

enumerator `ki2C_SlaveAddress3MatchFlag`
Slave address3 match. bit 21

enumerator `ki2C_MonitorReadyFlag`
The I2C monitor ready interrupt. bit 16

enumerator `ki2C_MonitorOverflowFlag`
The monitor data overrun interrupt. bit 17

enumerator `ki2C_MonitorActiveFlag`
The monitor is active. bit 18

enumerator `ki2C_MonitorIdleFlag`
The monitor idle interrupt. bit 19

enumerator `ki2C_EventTimeoutFlag`
The bus event timeout interrupt. bit 24

enumerator kI2C_SclTimeoutFlag

The SCL timeout interrupt. bit 25

enumerator kI2C_MasterAllClearFlags

enumerator kI2C_SlaveAllClearFlags

enumerator kI2C_CommonAllClearFlags

enum _i2c_interrupt_enable

I2C interrupt enable.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2C_MasterPendingInterruptEnable

The I2C master communication pending interrupt.

enumerator kI2C_MasterArbitrationLostInterruptEnable

The I2C master arbitration lost interrupt.

enumerator kI2C_MasterStartStopErrorInterruptEnable

The I2C master start/stop timing error interrupt.

enumerator kI2C_SlavePendingInterruptEnable

The I2C slave communication pending interrupt.

enumerator kI2C_SlaveNotStretchingInterruptEnable

The I2C slave not stretching interrupt, deep-sleep mode can be entered only when this interrupt occurs.

enumerator kI2C_SlaveDeselectedInterruptEnable

The I2C slave deselection interrupt.

enumerator kI2C_MonitorReadyInterruptEnable

The I2C monitor ready interrupt.

enumerator kI2C_MonitorOverflowInterruptEnable

The monitor data overrun interrupt.

enumerator kI2C_MonitorIdleInterruptEnable

The monitor idle interrupt.

enumerator kI2C_EventTimeoutInterruptEnable

The bus event timeout interrupt.

enumerator kI2C_SclTimeoutInterruptEnable

The SCL timeout interrupt.

enumerator kI2C_MasterAllInterruptEnable

enumerator kI2C_SlaveAllInterruptEnable

enumerator kI2C_CommonAllInterruptEnable

I2C_RETRY_TIMES

Retry times for waiting flag.

I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK

Whether to ignore the nack signal of the last byte during master transmit.

I2C_STAT_MSTCODE_IDLE

Master Idle State Code

I2C_STAT_MSTCODE_RXREADY

Master Receive Ready State Code

I2C_STAT_MSTCODE_TXREADY

Master Transmit Ready State Code

I2C_STAT_MSTCODE_NACKADR

Master NACK by slave on address State Code

I2C_STAT_MSTCODE_NACKDAT

Master NACK by slave on data State Code

I2C_STAT_SLVST_ADDR

I2C_STAT_SLVST_RX

I2C_STAT_SLVST_TX

2.38 I2C Master Driver

`void I2C_MasterGetDefaultConfig(i2c_master_config_t *masterConfig)`

Provides a default configuration for the I2C master peripheral.

This function provides the following default configuration for the I2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->baudRate_Bps      = 100000U;
masterConfig->enableTimeout     = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I2C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i2c_master_config_t`.

`void I2C_MasterInit(I2C_Type *base, const i2c_master_config_t *masterConfig, uint32_t srcClock_Hz)`

Initializes the I2C master peripheral.

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The I2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `srcClock_Hz` – Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`void I2C_MasterDeinit(I2C_Type *base)`

Deinitializes the I2C master peripheral.

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The I2C peripheral base address.

`uint32_t I2C_GetInstance(I2C_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 I2C peripheral base address.

Returns

I2C instance number starting from 0.

`static inline void I2C_MasterReset(I2C_Type *base)`

Performs a software reset.

Restores the I2C master peripheral to reset conditions.

Parameters

- `base` – The I2C peripheral base address.

`static inline void I2C_MasterEnable(I2C_Type *base, bool enable)`

Enables or disables the I2C module as master.

Parameters

- `base` – The I2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified I2C as master.

`uint32_t I2C_GetStatusFlags(I2C_Type *base)`

Gets the I2C status flags.

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i2c_status_flags`.

Parameters

- `base` – The I2C 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 I2C_ClearStatusFlags(I2C_Type *base, uint32_t statusMask)`

Clears the I2C status flag state.

Refer to `kI2C_CommonAllClearStatusFlags`, `kI2C_MasterAllClearStatusFlags` and `kI2C_SlaveAllClearStatusFlags` to see the clearable flags. Attempts to clear other flags has no effect.

See also:

`_i2c_status_flags`, `_i2c_master_status_flags` and `_i2c_slave_status_flags`.

Parameters

- `base` – The I2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of the members in `kI2C_CommonAllClearStatusFlags`, `kI2C_MasterAllClearStatusFlags` and `kI2C_SlaveAllClearStatusFlags`. You may pass the result of a previous call to `I2C_GetStatusFlags()`.

```
static inline void I2C_MasterClearStatusFlags(I2C_Type *base, uint32_t statusMask)
```

Clears the I2C master status flag state.

Deprecated:

Do not use this function. It has been superseded by `I2C_ClearStatusFlags`. The following status register flags can be cleared:

- `kI2C_MasterArbitrationLostFlag`
- `kI2C_MasterStartStopErrorFlag`

Attempts to clear other flags has no effect.

See also:

`_i2c_status_flags`.

Parameters

- `base` – The I2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i2c_status_flags` enumerators OR'd together. You may pass the result of a previous call to `I2C_GetStatusFlags()`.

```
static inline void I2C_EnableInterrupts(I2C_Type *base, uint32_t interruptMask)
```

Enables the I2C interrupt requests.

Parameters

- `base` – The I2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i2c_interrupt_enable` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void I2C_DisableInterrupts(I2C_Type *base, uint32_t interruptMask)
```

Disables the I2C interrupt requests.

Parameters

- `base` – The I2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_i2c_interrupt_enable` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I2C_GetEnabledInterrupts(I2C_Type *base)
```

Returns the set of currently enabled I2C interrupt requests.

Parameters

- `base` – The I2C peripheral base address.

Returns

A bitmask composed of `_i2c_interrupt_enable` enumerators OR'd together to indicate the set of enabled interrupts.

`void I2C_MasterSetBaudRate(I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)`

Sets the I2C bus frequency for master transactions.

The I2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

- `base` – The I2C peripheral base address.
- `srcClock_Hz` – I2C functional clock frequency in Hertz.
- `baudRate_Bps` – Requested bus frequency in bits per second.

`void I2C_MasterSetTimeoutValue(I2C_Type *base, uint8_t timeout_Ms, uint32_t srcClock_Hz)`

Sets the I2C bus timeout value.

If the SCL signal remains low or bus does not have event longer than the timeout value, `kI2C_SclTimeoutFlag` or `kI2C_EventTimeoutFlag` is set. This can indicate the bus is held by slave or any fault occurs to the I2C module.

Parameters

- `base` – The I2C peripheral base address.
- `timeout_Ms` – Timeout value in millisecond.
- `srcClock_Hz` – I2C functional clock frequency in Hertz.

`static inline bool I2C_MasterGetBusIdleState(I2C_Type *base)`

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The I2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

`status_t I2C_MasterStart(I2C_Type *base, uint8_t address, i2c_direction_t direction)`

Sends a START on the I2C bus.

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

- `base` – I2C peripheral base pointer
- `address` – 7-bit slave device address.
- `direction` – Master transfer directions(transmit/receive).

Return values

- `kStatus_Success` – Successfully send the start signal.
- `kStatus_I2C_Busy` – Current bus is busy.

status_t I2C_MasterStop(I2C_Type *base)

Sends a STOP signal on the I2C bus.

Return values

- kStatus_Success – Successfully send the stop signal.
- kStatus_I2C_Timeout – Send stop signal failed, timeout.

static inline *status_t* I2C_MasterRepeatedStart(I2C_Type *base, uint8_t address, *i2c_direction_t* direction)

Sends a REPEATED START on the I2C bus.

Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.
- direction – Master transfer directions(transmit/receive).

Return values

- kStatus_Success – Successfully send the start signal.
- kStatus_I2C_Busy – Current bus is busy but not occupied by current I2C master.

status_t I2C_MasterWriteBlocking(I2C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)

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_I2C_Nak.

Parameters

- base – The I2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- flags – Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_I2C_Busy – Another master is currently utilizing the bus.
- kStatus_I2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_I2C_ArbitrationLost – Arbitration lost error.

status_t I2C_MasterReadBlocking(I2C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The I2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- flags – Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_I2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_I2C_ArbitrationLost` – Arbitration lost error.

`status_t I2C_MasterTransferBlocking(I2C_Type *base, i2c_master_transfer_t *xfer)`

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

- `base` – I2C peripheral base address.
- `xfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Successfully complete the data transmission.
- `kStatus_I2C_Busy` – Previous transmission still not finished.
- `kStatus_I2C_Timeout` – Transfer error, wait signal timeout.
- `kStatus_I2C_ArbitrationLost` – Transfer error, arbitration lost.
- `kStatus_I2C_Nak` – Transfer error, receive NAK during transfer.
- `kStatus_I2C_Addr_Nak` – Transfer error, receive NAK during addressing.

`void I2C_MasterTransferCreateHandle(I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_callback_t callback, void *userData)`

Creates a new handle for the I2C 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 `I2C_MasterTransferAbort()` API shall be called.

Parameters

- `base` – The I2C peripheral base address.
- `handle` – **[out]** Pointer to the I2C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I2C_MasterTransferNonBlocking(I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_t *xfer)`

Performs a non-blocking transaction on the I2C bus.

Parameters

- `base` – The I2C peripheral base address.
- `handle` – Pointer to the I2C master driver handle.
- `xfer` – The pointer to the transfer descriptor.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I2C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

`status_t I2C_MasterTransferGetCount(I2C_Type *base, i2c_master_handle_t *handle, size_t *count)`

Returns number of bytes transferred so far.

Parameters

- base – The I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_I2C_Busy –

`status_t I2C_MasterTransferAbort(I2C_Type *base, i2c_master_handle_t *handle)`

Terminates a non-blocking I2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

Parameters

- base – The I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_I2C_Timeout – Timeout during polling for flags.

`void I2C_MasterTransferHandleIRQ(I2C_Type *base, i2c_master_handle_t *handle)`

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 I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.

`enum _i2c_direction`

Direction of master and slave transfers.

Values:

enumerator kI2C_Write

Master transmit.

enumerator kI2C_Read

Master receive.

enum `_i2c_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 `_i2c_master_transfer::flags` field.

Values:

enumerator `kI2C_TransferDefaultFlag`

Transfer starts with a start signal, stops with a stop signal.

enumerator `kI2C_TransferNoStartFlag`

Don't send a start condition, address, and sub address

enumerator `kI2C_TransferRepeatedStartFlag`

Send a repeated start condition

enumerator `kI2C_TransferNoStopFlag`

Don't send a stop condition.

enum `_i2c_transfer_states`

States for the state machine used by transactional APIs.

Values:

enumerator `kIdleState`

enumerator `kTransmitSubaddrState`

enumerator `kTransmitDataState`

enumerator `kReceiveDataBeginState`

enumerator `kReceiveDataState`

enumerator `kReceiveLastDataState`

enumerator `kStartState`

enumerator `kStopState`

enumerator `kWaitForCompletionState`

typedef enum `_i2c_direction` `i2c_direction_t`

Direction of master and slave transfers.

typedef struct `_i2c_master_config` `i2c_master_config_t`

Structure with settings to initialize the I2C master module.

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the `I2C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef struct `_i2c_master_transfer` `i2c_master_transfer_t`

I2C master transfer typedef.

typedef struct `_i2c_master_handle` `i2c_master_handle_t`

I2C master handle typedef.


```
typedef void (*i2c_master_transfer_callback_t)(I2C_Type *base, i2c_master_handle_t *handle,
status_t 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 I2C_MasterTransferCreateHandle().

Param base

The I2C peripheral base address.

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 _i2c_master_config
```

#include <fsl_i2c.h> Structure with settings to initialize the I2C master module.

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the I2C_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.

uint32_t baudRate_Bps

Desired baud rate in bits per second.

bool enableTimeout

Enable internal timeout function.

uint8_t timeout_Ms

Event timeout and SCL low timeout value.

```
struct _i2c_master_transfer
```

#include <fsl_i2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration i2c_master_transfer_flags for available options. Set to 0 or kI2C_TransferDefaultFlag for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i2c_direction_t direction

Either kI2C_Read or kI2C_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 _i2c_master_handle

#include <fsl_i2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint32_t transferCount

Indicates progress of the transfer

uint32_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

bool checkAddrNack

Whether to check the nack signal is detected during addressing.

i2c_master_transfer_t transfer

Copy of the current transfer info.

i2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

2.39 I2C Slave Driver

void I2C_SlaveGetDefaultConfig(*i2c_slave_config_t* *slaveConfig)

Provides a default configuration for the I2C slave peripheral.

This function provides the following default configuration for the I2C slave peripheral:

```
slaveConfig->enableSlave = true;
slaveConfig->address0.disable = false;
slaveConfig->address0.address = 0u;
slaveConfig->address1.disable = true;
slaveConfig->address2.disable = true;
slaveConfig->address3.disable = true;
slaveConfig->busSpeed = kI2C_SlaveStandardMode;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with I2C_SlaveInit(). Be sure to override at least the *address0.address* 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 `i2c_slave_config_t`.

`status_t I2C_SlaveInit(I2C_Type *base, const i2c_slave_config_t *slaveConfig, uint32_t srcClock_Hz)`

Initializes the I2C slave peripheral.

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

Parameters

- base – The I2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use `I2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- srcClock_Hz – Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock.

`void I2C_SlaveSetAddress(I2C_Type *base, i2c_slave_address_register_t addressRegister, uint8_t address, bool addressDisable)`

Configures Slave Address n register.

This function writes new value to Slave Address register.

Parameters

- base – The I2C peripheral base address.
- addressRegister – The module supports multiple address registers. The parameter determines which one shall be changed.
- address – The slave address to be stored to the address register for matching.
- addressDisable – Disable matching of the specified address register.

`void I2C_SlaveDeinit(I2C_Type *base)`

Deinitializes the I2C slave peripheral.

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The I2C peripheral base address.

`static inline void I2C_SlaveEnable(I2C_Type *base, bool enable)`

Enables or disables the I2C module as slave.

Parameters

- base – The I2C peripheral base address.
- enable – True to enable or false to disable.

`static inline void I2C_SlaveClearStatusFlags(I2C_Type *base, uint32_t statusMask)`

Clears the I2C status flag state.

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

See also:

`_i2c_slave_flags`.

Parameters

- `base` – The I2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I2C_SlaveGetStatusFlags()`.

status_t I2C_SlaveWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

The function executes blocking address phase and blocking data phase.

Parameters

- `base` – The I2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

`kStatus_Success` Data has been sent.

Returns

`kStatus_Fail_Unexpected_slave_state` (master data write while master read from slave is expected).

status_t I2C_SlaveReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

The function executes blocking address phase and blocking data phase.

Parameters

- `base` – The I2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Returns

`kStatus_Success` Data has been received.

Returns

`kStatus_Fail_Unexpected_slave_state` (master data read while master write to slave is expected).

void I2C_SlaveTransferCreateHandle(I2C_Type *base, *i2c_slave_handle_t* *handle, *i2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the I2C 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 `I2C_SlaveTransferAbort()` API shall be called.

Parameters

- `base` – The I2C peripheral base address.
- `handle` – **[out]** Pointer to the I2C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.

- `userData` – User provided pointer to the application callback data.

`status_t I2C_SlaveTransferNonBlocking(I2C_Type *base, i2c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I2C_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 `I2C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes `kI2C_SlaveTransmitEvent` callback. If no slave Rx transfer is busy, a master write to slave request invokes `kI2C_SlaveReceiveEvent` callback.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI2C_SlaveTransmitEvent` and `kI2C_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 `kI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

- `base` – The I2C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together `i2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI2C_SlaveAllEvents` to enable all events.

Return values

- `kStatus_Success` – Slave transfers were successfully started.
- `kStatus_I2C_Busy` – Slave transfers have already been started on this handle.

`status_t I2C_SlaveSetSendBuffer(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, const void *txData, size_t txSize, uint32_t eventMask)`

Starts accepting master read from slave requests.

The function can be called in response to `kI2C_SlaveTransmitEvent` callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI2C_SlaveTransmitEvent` and `kI2C_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 `kI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

- `base` – The I2C peripheral base address.
- `transfer` – Pointer to `i2c_slave_transfer_t` structure.
- `txData` – Pointer to data to send to master.
- `txSize` – Size of `txData` in bytes.

- `eventMask` – Bit mask formed by OR'ing together `i2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI2C_SlaveAllEvents` to enable all events.

Return values

- `kStatus_Success` – Slave transfers were successfully started.
- `kStatus_I2C_Busy` – Slave transfers have already been started on this handle.

`status_t I2C_SlaveSetReceiveBuffer(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *rxData, size_t rxSize, uint32_t eventMask)`

Starts accepting master write to slave requests.

The function can be called in response to `kI2C_SlaveReceiveEvent` callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI2C_SlaveTransmitEvent` and `kI2C_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 `kI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

- `base` – The I2C peripheral base address.
- `transfer` – Pointer to `i2c_slave_transfer_t` structure.
- `rxData` – Pointer to data to store data from master.
- `rxSize` – Size of `rxData` in bytes.
- `eventMask` – Bit mask formed by OR'ing together `i2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI2C_SlaveAllEvents` to enable all events.

Return values

- `kStatus_Success` – Slave transfers were successfully started.
- `kStatus_I2C_Busy` – Slave transfers have already been started on this handle.

`static inline uint32_t I2C_SlaveGetReceivedAddress(I2C_Type *base, volatile i2c_slave_transfer_t *transfer)`

Returns the slave address sent by the I2C master.

This function should only be called from the address match event callback `kI2C_SlaveAddressMatchEvent`.

Parameters

- `base` – The I2C peripheral base address.
- `transfer` – The I2C slave transfer.

Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

`void I2C_SlaveTransferAbort(I2C_Type *base, i2c_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 I2C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure which stores the transfer state.

Return values

- `kStatus_Success` –
- `kStatus_I2C_Idle` –

`status_t I2C_SlaveTransferGetCount(I2C_Type *base, i2c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

Parameters

- `base` – I2C base pointer.
- `handle` – pointer to `i2c_slave_handle_t` structure.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_Success` – Successfully return the count.

`void I2C_SlaveTransferHandleIRQ(I2C_Type *base, i2c_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 I2C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure which stores the transfer state.

`enum _i2c_slave_address_register`

I2C slave address register.

Values:

enumerator `kI2C_SlaveAddressRegister0`

Slave Address 0 register.

enumerator `kI2C_SlaveAddressRegister1`

Slave Address 1 register.

enumerator `kI2C_SlaveAddressRegister2`

Slave Address 2 register.

enumerator `kI2C_SlaveAddressRegister3`

Slave Address 3 register.

enum `_i2c_slave_address_qual_mode`

I2C slave address match options.

Values:

enumerator `ki2C_QualModeMask`

The SLVQUAL0 field (`qualAddress`) is used as a logical mask for matching address0.

enumerator `ki2C_QualModeExtend`

The SLVQUAL0 (`qualAddress`) field is used to extend address 0 matching in a range of addresses.

enum `_i2c_slave_bus_speed`

I2C slave bus speed options.

Values:

enumerator `ki2C_SlaveStandardMode`

enumerator `ki2C_SlaveFastMode`

enumerator `ki2C_SlaveFastModePlus`

enumerator `ki2C_SlaveHsMode`

enum `_i2c_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 `I2C_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 `ki2C_SlaveAddressMatchEvent`

Received the slave address after a start or repeated start.

enumerator `ki2C_SlaveTransmitEvent`

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator `ki2C_SlaveReceiveEvent`

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator `ki2C_SlaveCompletionEvent`

All data in the active transfer have been consumed.

enumerator `ki2C_SlaveDeselectedEvent`

The slave function has become deselected (SLVSEL flag changing from 1 to 0).

enumerator `ki2C_SlaveAllEvents`

Bit mask of all available events.

enum `_i2c_slave_fsm`

I2C slave software finite state machine states.

Values:

enumerator `ki2C_SlaveFsmAddressMatch`

enumerator `kI2C_SlaveFsmReceive`

enumerator `kI2C_SlaveFsmTransmit`

typedef enum `_i2c_slave_address_register` `i2c_slave_address_register_t`
I2C slave address register.

typedef struct `_i2c_slave_address` `i2c_slave_address_t`
Data structure with 7-bit Slave address and Slave address disable.

typedef enum `_i2c_slave_address_qual_mode` `i2c_slave_address_qual_mode_t`
I2C slave address match options.

typedef enum `_i2c_slave_bus_speed` `i2c_slave_bus_speed_t`
I2C slave bus speed options.

typedef struct `_i2c_slave_config` `i2c_slave_config_t`
Structure with settings to initialize the I2C slave module.

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the `I2C_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 `_i2c_slave_transfer_event` `i2c_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 `I2C_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 `_i2c_slave_handle` `i2c_slave_handle_t`
I2C slave handle typedef.

typedef struct `_i2c_slave_transfer` `i2c_slave_transfer_t`
I2C slave transfer structure.

typedef void (*`i2c_slave_transfer_callback_t`)(`I2C_Type` *`base`, volatile `i2c_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 `I2C_SlaveSetCallback()` function after you have created a handle.

Param base

Base address for the I2C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

typedef enum `_i2c_slave_fsm` `i2c_slave_fsm_t`
I2C slave software finite state machine states.

```
typedef void (*flexcomm_i2c_master_irq_handler_t)(I2C_Type *base, i2c_master_handle_t *handle)
```

Typedef for master interrupt handler.

```
typedef void (*flexcomm_i2c_slave_irq_handler_t)(I2C_Type *base, i2c_slave_handle_t *handle)
```

Typedef for slave interrupt handler.

```
struct _i2c_slave_address
```

#include <fsl_i2c.h> Data structure with 7-bit Slave address and Slave address disable.

Public Members

uint8_t address

7-bit Slave address SLVADR.

bool addressDisable

Slave address disable SADISABLE.

```
struct _i2c_slave_config
```

#include <fsl_i2c.h> Structure with settings to initialize the I2C slave module.

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the I2C_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

i2c_slave_address_t address0

Slave's 7-bit address and disable.

i2c_slave_address_t address1

Alternate slave 7-bit address and disable.

i2c_slave_address_t address2

Alternate slave 7-bit address and disable.

i2c_slave_address_t address3

Alternate slave 7-bit address and disable.

i2c_slave_address_qual_mode_t qualMode

Qualify mode for slave address 0.

uint8_t qualAddress

Slave address qualifier for address 0.

i2c_slave_bus_speed_t busSpeed

Slave bus speed mode. If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The busSpeed value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the busSpeed mode is unknown at compile time, use the longest data setup time kI2C_SlaveStandardMode (250 ns)

bool enableSlave

Enable slave mode.

```
struct _i2c_slave_transfer
```

#include <fsl_i2c.h> I2C slave transfer structure.

Public Members

i2c_slave_handle_t *handle

Pointer to handle that contains this transfer.

i2c_slave_transfer_event_t event

Reason the callback is being invoked.

uint8_t receivedAddress

Matching address send by master. 7-bits plus R/nW bit0

uint32_t eventMask

Mask of enabled events.

uint8_t *rxData

Transfer buffer for receive data

const uint8_t *txData

Transfer buffer for transmit data

size_t txSize

Transfer size

size_t rxSize

Transfer size

size_t transferredCount

Number of bytes transferred during this transfer.

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI2C_SlaveCompletionEvent.

struct *_i2c_slave_handle*

#include <fsl_i2c.h> I2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

volatile *i2c_slave_transfer_t* transfer

I2C slave transfer.

volatile bool isBusy

Whether transfer is busy.

volatile *i2c_slave_fsm_t* slaveFsm

slave transfer state machine.

i2c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

2.40 I2S: I2S Driver

2.41 I2S DMA Driver

```
void I2S_TxTransferCreateHandleDMA(I2S_Type *base, i2s_dma_handle_t *handle, dma_handle_t *dmaHandle, i2s_dma_transfer_callback_t callback, void *userData)
```

Initializes handle for transfer of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- dmaHandle – pointer to dma handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

```
status_t I2S_TxTransferSendDMA(I2S_Type *base, i2s_dma_handle_t *handle, i2s_transfer_t transfer)
```

Begins or queue sending of the given data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with unsent buffers.

```
void I2S_TransferAbortDMA(I2S_Type *base, i2s_dma_handle_t *handle)
```

Aborts transfer of data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

```
void I2S_RxTransferCreateHandleDMA(I2S_Type *base, i2s_dma_handle_t *handle, dma_handle_t *dmaHandle, i2s_dma_transfer_callback_t callback, void *userData)
```

Initializes handle for reception of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- dmaHandle – pointer to dma handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

```
status_t I2S_RxTransferReceiveDMA(I2S_Type *base, i2s_dma_handle_t *handle, i2s_transfer_t transfer)
```

Begins or queue reception of data into given buffer.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with buffers which are not full.

```
void I2S_DMACallback(dma_handle_t *handle, void *userData, bool transferDone, uint32_t tcds)
```

Invoked from DMA interrupt handler.

Parameters

- handle – pointer to DMA handle structure.
- userData – argument for user callback.
- transferDone – if transfer was done.
- tcds –

```
void I2S_TransferInstallLoopDMADescriptorMemory(i2s_dma_handle_t *handle, void *dmaDescriptorAddr, size_t dmaDescriptorNum)
```

Install DMA descriptor memory for loop transfer only.

This function used to register DMA descriptor memory for the i2s loop dma transfer.

It must be called before I2S_TransferSendLoopDMA/I2S_TransferReceiveLoopDMA and after I2S_RxTransferCreateHandleDMA/I2S_TxTransferCreateHandleDMA.

User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

- handle – Pointer to i2s DMA transfer handle.
- dmaDescriptorAddr – DMA descriptor start address.
- dmaDescriptorNum – DMA descriptor number.

```
status_t I2S_TransferSendLoopDMA(I2S_Type *base, i2s_dma_handle_t *handle, i2s_transfer_t *xfer, uint32_t loopTransferCount)
```

Send link transfer data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S_InstallDMADescriptorMemory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S_TransferAbortDMA to stop the loop transfer.

Parameters

- base – I2S peripheral base address.
- handle – Pointer to `usart_dma_handle_t` structure.
- xfer – I2S DMA transfer structure. See `i2s_transfer_t`.
- loopTransferCount – loop count

Return values

`kStatus_Success` –

`status_t` I2S_TransferReceiveLoopDMA(I2S_Type *base, *i2s_dma_handle_t* *handle, *i2s_transfer_t* *xfer, uint32_t loopTransferCount)

Receive link transfer data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function `I2S_InstallDMADescriptorMemory` to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function `I2S_TransferAbortDMA` to stop the loop transfer.

Parameters

- base – I2S peripheral base address.
- handle – Pointer to `usart_dma_handle_t` structure.
- xfer – I2S DMA transfer structure. See `i2s_transfer_t`.
- loopTransferCount – loop count

Return values

`kStatus_Success` –

FSL_I2S_DMA_DRIVER_VERSION

I2S DMA driver version 2.3.3.

`typedef struct _i2s_dma_handle` `i2s_dma_handle_t`

Members not to be accessed / modified outside of the driver.

`typedef void (*i2s_dma_transfer_callback_t)(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData)`

Callback function invoked from DMA API on completion.

Param base

I2S base pointer.

Param handle

pointer to I2S transaction.

Param completionStatus

status of the transaction.

Param userData

optional pointer to user arguments data.

`struct _i2s_dma_handle`

`#include <fsl_i2s_dma.h>` i2s dma handle

Public Members

`uint32_t` state
Internal state of I2S DMA transfer

`uint8_t` bytesPerFrame
bytes per frame

`i2s_dma_transfer_callback_t` completionCallback
Callback function pointer

`void *`userData
Application data passed to callback

`dma_handle_t *`dmaHandle
DMA handle

`volatile i2s_transfer_t` i2sQueue[(4U)]
Transfer queue storing transfer buffers

`volatile uint8_t` queueUser
Queue index where user's next transfer will be stored

`volatile uint8_t` queueDriver
Queue index of buffer actually used by the driver

`dma_descriptor_t *`i2sLoopDMADescriptor
descriptor pool pointer

`size_t` i2sLoopDMADescriptorNum
number of descriptor in descriptors pool

2.42 I2S Driver

`void` I2S_TxInit(I2S_Type *base, const `i2s_config_t` *config)

Initializes the FLEXCOMM peripheral for I2S transmit functionality.

Ungates the FLEXCOMM clock and configures the module for I2S transmission using a configuration structure. The configuration structure can be custom filled or set with default values by I2S_TxGetDefaultConfig().

Note: This API should be called at the beginning of the application to use the I2S driver.

Parameters

- base – I2S base pointer.
- config – pointer to I2S configuration structure.

`void` I2S_RxInit(I2S_Type *base, const `i2s_config_t` *config)

Initializes the FLEXCOMM peripheral for I2S receive functionality.

Ungates the FLEXCOMM clock and configures the module for I2S receive using a configuration structure. The configuration structure can be custom filled or set with default values by I2S_RxGetDefaultConfig().

Note: This API should be called at the beginning of the application to use the I2S driver.

Parameters

- base – I2S base pointer.
- config – pointer to I2S configuration structure.

`void I2S_TxGetDefaultConfig(i2s_config_t *config)`

Sets the I2S Tx configuration structure to default values.

This API initializes the configuration structure for use in `I2S_TxInit()`. The initialized structure can remain unchanged in `I2S_TxInit()`, or it can be modified before calling `I2S_TxInit()`. Example:

```
i2s_config_t config;
I2S_TxGetDefaultConfig(&config);
```

Default values:

```
config->masterSlave = kI2S_MasterSlaveNormalMaster;
config->mode = kI2S_ModeI2sClassic;
config->rightLow = false;
config->leftJust = false;
config->pdmData = false;
config->sckPol = false;
config->wsPol = false;
config->divider = 1;
config->oneChannel = false;
config->dataLength = 16;
config->frameLength = 32;
config->position = 0;
config->watermark = 4;
config->txEmptyZero = true;
config->pack48 = false;
```

Parameters

- config – pointer to I2S configuration structure.

`void I2S_RxGetDefaultConfig(i2s_config_t *config)`

Sets the I2S Rx configuration structure to default values.

This API initializes the configuration structure for use in `I2S_RxInit()`. The initialized structure can remain unchanged in `I2S_RxInit()`, or it can be modified before calling `I2S_RxInit()`. Example:

```
i2s_config_t config;
I2S_RxGetDefaultConfig(&config);
```

Default values:

```
config->masterSlave = kI2S_MasterSlaveNormalSlave;
config->mode = kI2S_ModeI2sClassic;
config->rightLow = false;
config->leftJust = false;
config->pdmData = false;
config->sckPol = false;
config->wsPol = false;
config->divider = 1;
config->oneChannel = false;
config->dataLength = 16;
config->frameLength = 32;
config->position = 0;
config->watermark = 4;
```

(continues on next page)

(continued from previous page)

```
config->txEmptyZero = false;
config->pack48 = false;
```

Parameters

- config – pointer to I2S configuration structure.

void I2S_Deinit(I2S_Type *base)

De-initializes the I2S peripheral.

This API gates the FLEXCOMM clock. The I2S module can't operate unless I2S_TxInit or I2S_RxInit is called to enable the clock.

Parameters

- base – I2S base pointer.

void I2S_SetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter/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 I2S_TxTransferCreateHandle(I2S_Type *base, *i2s_handle_t* *handle, *i2s_transfer_callback_t* callback, void *userData)

Initializes handle for transfer of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

status_t I2S_TxTransferNonBlocking(I2S_Type *base, *i2s_handle_t* *handle, *i2s_transfer_t* transfer)

Begins or queue sending of the given data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with unsent buffers.

`void I2S_TxTransferAbort(I2S_Type *base, i2s_handle_t *handle)`

Aborts sending of data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

`void I2S_RxTransferCreateHandle(I2S_Type *base, i2s_handle_t *handle, i2s_transfer_callback_t callback, void *userData)`

Initializes handle for reception of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

`status_t I2S_RxTransferNonBlocking(I2S_Type *base, i2s_handle_t *handle, i2s_transfer_t transfer)`

Begins or queue reception of data into given buffer.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with buffers which are not full.

`void I2S_RxTransferAbort(I2S_Type *base, i2s_handle_t *handle)`

Aborts receiving of data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

`status_t I2S_TransferGetCount(I2S_Type *base, i2s_handle_t *handle, size_t *count)`

Returns number of bytes transferred so far.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- count – **[out]** number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – there is no non-blocking transaction currently in progress.

status_t I2S_TransferGetErrorCount(I2S_Type *base, *i2s_handle_t* *handle, *size_t* *count)

Returns number of buffer underruns or overruns.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- count – **[out]** number of transmit errors encountered so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – there is no non-blocking transaction currently in progress.

static inline void I2S_Enable(I2S_Type *base)

Enables I2S operation.

Parameters

- base – I2S base pointer.

void I2S_EnableSecondaryChannel(I2S_Type *base, *uint32_t* channel, *bool* oneChannel, *uint32_t* position)

Enables I2S secondary channel.

Parameters

- base – I2S base pointer.
- channel – secondary channel number, reference `_i2s_secondary_channel`.
- oneChannel – true is treated as single channel, functionality left channel for this pair.
- position – define the location within the frame of the data, should not bigger than 0x1FFU.

static inline void I2S_DisableSecondaryChannel(I2S_Type *base, *uint32_t* channel)

Disables I2S secondary channel.

Parameters

- base – I2S base pointer.
- channel – secondary channel number, reference `_i2s_secondary_channel`.

static inline void I2S_Disable(I2S_Type *base)

Disables I2S operation.

Parameters

- base – I2S base pointer.

static inline void I2S_EnableInterrupts(I2S_Type *base, *uint32_t* interruptMask)

Enables I2S FIFO interrupts.

Parameters

- base – I2S base pointer.
- interruptMask – bit mask of interrupts to enable. See `i2s_flags_t` for the set of constants that should be OR'd together to form the bit mask.

static inline void I2S_DisableInterrupts(I2S_Type *base, uint32_t interruptMask)

Disables I2S FIFO interrupts.

Parameters

- base – I2S base pointer.
- interruptMask – bit mask of interrupts to enable. See `i2s_flags_t` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I2S_GetEnabledInterrupts(I2S_Type *base)

Returns the set of currently enabled I2S FIFO interrupts.

Parameters

- base – I2S base pointer.

Returns

A bitmask composed of `i2s_flags_t` enumerators OR'd together to indicate the set of enabled interrupts.

status_t I2S_EmptyTxFifo(I2S_Type *base)

Flush the valid data in TX fifo.

Parameters

- base – I2S base pointer.

Returns

`kStatus_Fail` empty TX fifo failed, `kStatus_Success` empty tx fifo success.

void I2S_TxHandleIRQ(I2S_Type *base, i2s_handle_t *handle)

Invoked from interrupt handler when transmit FIFO level decreases.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

void I2S_RxHandleIRQ(I2S_Type *base, i2s_handle_t *handle)

Invoked from interrupt handler when receive FIFO level decreases.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

FSL_I2S_DRIVER_VERSION

I2S driver version 2.3.2.

`_i2s_status` I2S status codes.

Values:

enumerator `kStatus_I2S_BufferComplete`

Transfer from/into a single buffer has completed

enumerator `kStatus_I2S_Done`

All buffers transfers have completed

enumerator `kStatus_I2S_Busy`

Already performing a transfer and cannot queue another buffer

enum _i2s_flags

I2S flags.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2S_TxErrorFlag

TX error interrupt

enumerator kI2S_TxLevelFlag

TX level interrupt

enumerator kI2S_RxErrorFlag

RX error interrupt

enumerator kI2S_RxLevelFlag

RX level interrupt

enum _i2s_master_slave

Master / slave mode.

Values:

enumerator kI2S_MasterSlaveNormalSlave

Normal slave

enumerator kI2S_MasterSlaveWsSyncMaster

WS synchronized master

enumerator kI2S_MasterSlaveExtSckMaster

Master using existing SCK

enumerator kI2S_MasterSlaveNormalMaster

Normal master

enum _i2s_mode

I2S mode.

Values:

enumerator kI2S_ModeI2sClassic

I2S classic mode

enumerator kI2S_ModeDspWs50

DSP mode, WS having 50% duty cycle

enumerator kI2S_ModeDspWsShort

DSP mode, WS having one clock long pulse

enumerator kI2S_ModeDspWsLong

DSP mode, WS having one data slot long pulse

_i2s_secondary_channel I2S secondary channel.

Values:

enumerator kI2S_SecondaryChannel1

secondary channel 1

enumerator `kI2S_SecondaryChannel2`
secondary channel 2

enumerator `kI2S_SecondaryChannel3`
secondary channel 3

typedef enum `_i2s_flags` `i2s_flags_t`
I2S flags.

Note: These enums are meant to be OR'd together to form a bit mask.

typedef enum `_i2s_master_slave` `i2s_master_slave_t`
Master / slave mode.

typedef enum `_i2s_mode` `i2s_mode_t`
I2S mode.

typedef struct `_i2s_config` `i2s_config_t`
I2S configuration structure.

typedef struct `_i2s_transfer` `i2s_transfer_t`
Buffer to transfer from or receive audio data into.

typedef struct `_i2s_handle` `i2s_handle_t`
Transactional state of the initialized transfer or receive I2S operation.

typedef void (`*i2s_transfer_callback_t`)(`I2S_Type *base`, `i2s_handle_t *handle`, `status_t` `completionStatus`, void `*userData`)

Callback function invoked from transactional API on completion of a single buffer transfer.

Param base
I2S base pointer.

Param handle
pointer to I2S transaction.

Param completionStatus
status of the transaction.

Param userData
optional pointer to user arguments data.

`I2S_NUM_BUFFERS`
Number of buffers .

struct `_i2s_config`
`#include <fsl_i2s.h>` I2S configuration structure.

Public Members

`i2s_master_slave_t` `masterSlave`
Master / slave configuration

`i2s_mode_t` `mode`
I2S mode

bool `rightLow`
Right channel data in low portion of FIFO

`bool leftJust`
Left justify data in FIFO

`bool pdmData`
Data source is the D-Mic subsystem

`bool sckPol`
SCK polarity

`bool wsPol`
WS polarity

`uint16_t divider`
Flexcomm function clock divider (1 - 4096)

`bool oneChannel`
true mono, false stereo

`uint8_t dataLength`
Data length (4 - 32)

`uint16_t frameLength`
Frame width (4 - 512)

`uint16_t position`
Data position in the frame

`uint8_t watermark`
FIFO trigger level

`bool txEmptyZero`
Transmit zero when buffer becomes empty or last item

`bool pack48`
Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

`struct _i2s_transfer`
#include <fsl_i2s.h> Buffer to transfer from or receive audio data into.

Public Members

`uint8_t *data`
Pointer to data buffer.

`size_t dataSize`
Buffer size in bytes.

`struct _i2s_handle`
#include <fsl_i2s.h> Members not to be accessed / modified outside of the driver.

Public Members

`volatile uint32_t state`
State of transfer

`i2s_transfer_callback_t completionCallback`
Callback function pointer

`void *userData`
Application data passed to callback

`bool oneChannel`
true mono, false stereo

`uint8_t dataLength`
Data length (4 - 32)

`bool pack48`
Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

`uint8_t watermark`
FIFO trigger level

`bool useFifo48H`
When dataLength 17-24: true use FIFOWR48H, false use FIFOWR

`volatile i2s_transfer_t i2sQueue[(4U)]`
Transfer queue storing transfer buffers

`volatile uint8_t queueUser`
Queue index where user's next transfer will be stored

`volatile uint8_t queueDriver`
Queue index of buffer actually used by the driver

`volatile uint32_t errorCount`
Number of buffer underruns/overruns

`volatile uint32_t transferCount`
Number of bytes transferred

2.43 I3C: I3C Driver

`FSL_I3C_DRIVER_VERSION`
I3C driver version.

I3C status return codes.

Values:

`enumerator kStatus_I3C_Busy`
The master is already performing a transfer.

`enumerator kStatus_I3C_Idle`
The slave driver is idle.

`enumerator kStatus_I3C_Nak`
The slave device sent a NAK in response to an address.

`enumerator kStatus_I3C_WriteAbort`
The slave device sent a NAK in response to a write.

`enumerator kStatus_I3C_Term`
The master terminates slave read.

enumerator kStatus_I3C_HdrParityError
Parity error from DDR read.

enumerator kStatus_I3C_CrcError
CRC error from DDR read.

enumerator kStatus_I3C_ReadFifoError
Read from M/SRDATA register when FIFO empty.

enumerator kStatus_I3C_WriteFifoError
Write to M/SWDATA register when FIFO full.

enumerator kStatus_I3C_MsgError
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus_I3C_InvalidReq
Invalid use of request.

enumerator kStatus_I3C_Timeout
The module has stalled too long in a frame.

enumerator kStatus_I3C_SlaveCountExceed
The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

enumerator kStatus_I3C_IBIWon
The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus_I3C_OverrunError
Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus_I3C_UnderrunError
Slave internal to-bus buffer/FIFO underrun

enumerator kStatus_I3C_UnderrunNak
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus_I3C_InvalidStart
Slave invalid start flag

enumerator kStatus_I3C_SdrParityError
SDR parity error

enumerator kStatus_I3C_S0S1Error
S0 or S1 error

enum _i3c_hdr_mode
I3C HDR modes.
Values:

enumerator kI3C_HDRModeNone

enumerator kI3C_HDRModeDDR

enumerator kI3C_HDRModeTSP

enumerator kI3C_HDRModeTSL

typedef enum _i3c_hdr_mode i3c_hdr_mode_t
I3C HDR modes.

typedef struct _i3c_device_info i3c_device_info_t
I3C device information.

I3C_RETRY_TIMES

Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C_MAX_DEVCNT

I3C_IBI_BUFF_SIZE

struct _i3c_device_info

#include <fsl_i3c.h> I3C device information.

Public Members

uint8_t dynamicAddr

Device dynamic address.

uint8_t staticAddr

Static address.

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint16_t maxReadLength

Maximum read length.

uint16_t maxWriteLength

Maximum write length.

uint8_t hdrMode

Support hdr mode, could be OR logic in i3c_hdr_mode.

2.44 I3C Common Driver

typedef struct _i3c_config i3c_config_t

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

uint32_t I3C_GetInstance(I3C_Type *base)

Get which instance current I3C is used.

Parameters

- base – The I3C peripheral base address.

`void I3C_GetDefaultConfig(i3c_config_t *config)`

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster          = kI3C_MasterCapable;
config->disableTimeout        = false;
config->hKeep                  = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop    = true;
config->enableOpenDrainHigh    = true;
config->baudRate_Hz.i2cBaud    = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress    = 0x0AU;
config->slowClock_Hz           = 1000000U;
config->enableSlave            = true;
config->vendorID                = 0x11BU;
config->enableRandomPart       = false;
config->partNumber              = 0;
config->dcr                     = 0;
config->bcr = 0;
config->hdrMode                 = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest           = false;
config->ignoreS0S1Error        = false;
config->offline                 = false;
config->matchSlaveStartStop    = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with `I3C_Init()`.

Parameters

- `config` – **[out]** User provided configuration structure for default values. Refer to `i3c_config_t`.

`void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)`

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- `base` – The I3C peripheral base address.
- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`struct _i3c_config`

`#include <fsl_i3c.h>` Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

uint8_t masterDynamicAddress

Main master dynamic address configuration.

uint32_t maxWriteLength

Maximum write length.

uint32_t maxReadLength

Maximum read length.

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint8_t hdrMode

Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

2.45 I3C Master DMA Driver

```
typedef struct _i3c_master_dma_handle i3c_master_dma_handle_t
```

```
typedef struct _i3c_master_dma_callback i3c_master_dma_callback_t
    i3c master callback functions.
```

```
void I3C_MasterTransferCreateHandleDMA(I3C_Type *base, i3c_master_dma_handle_t *handle,
    const i3c_master_dma_callback_t *callback, void
    *userData, dma_handle_t *rxDmaHandle,
    dma_handle_t *txDmaHandle)
```

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.
- *rxDmaHandle* – Handle for the DMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_MasterTransferDMA(I3C_Type *base, i3c_master_dma_handle_t *handle,
    i3c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t I3C_MasterTransferGetCountDMA(I3C_Type *base, i3c_master_dma_handle_t *handle,
    size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.

- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` – There is not a DMA transaction currently in progress.

`void I3C_MasterTransferAbortDMA(I3C_Type *base, i3c_master_dma_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferDMAHandleIRQ(I3C_Type *base, void *i3CHandle)`

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master DMA driver handle.

`void (*slave2Master)(I3C_Type *base, void *userData)`

Transfer complete callback

`void (*ibiCallback)(I3C_Type *base, i3c_master_dma_handle_t *handle, i3c_ibi_type_t ibiType, i3c_ibi_state_t ibiState)`

IBI event callback

`void (*transferComplete)(I3C_Type *base, i3c_master_dma_handle_t *handle, status_t status, void *userData)`

Transfer complete callback

`I3C_Type *base`

I3C base pointer.

`uint8_t state`

Transfer state machine current state.

`uint32_t transferCount`

Indicates progress of the transfer

`uint8_t subaddressBuffer[4]`

Saving subaddress command.

`uint8_t subaddressCount`

Saving command count.

i3c_master_transfer_t transfer

Copy of the current transfer info.

i3c_master_dma_callback_t callback

Callback function pointer.

void *userData

Application data passed to callback.

dma_handle_t *rxDmaHandle

Handle for receive DMA channel.

dma_handle_t *txDmaHandle

Handle for transmit DMA channel.

uint8_t ibiAddress

Slave address which request IBI.

uint8_t *ibiBuff

Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize

IBI payload size.

i3c_ibi_type_t ibiType

IBI type.

uint32_t event

Reason the callback is being invoked.

uint8_t *txData

Transfer buffer

size_t txDataSize

Transfer size

uint8_t *rxData

Transfer buffer

size_t rxDataSize

Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI3C_SlaveCompletionEvent.

I3C_Type *base

I3C base pointer.

i3c_slave_dma_transfer_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

i3c_slave_dma_callback_t callback

Callback function called at transfer event.

dma_handle_t *rxDmaHandle

Handle for receive DMA channel.

dma_handle_t *txDmaHandle

Handle for transmit DMA channel.

void *userData

Callback parameter passed to callback.

struct *_i3c_master_dma_callback*

#include <fsl_i3c_dma.h> i3c master callback functions.

struct *_i3c_master_dma_handle*

#include <fsl_i3c_dma.h> Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

2.46 I3C Master Driver

void I3C_MasterGetDefaultConfig(*i3c_master_config_t* *masterConfig)

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster      = kI3C_MasterOn;
masterConfig->disableTimeout    = false;
masterConfig->hKeep              = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop = true;
masterConfig->enableOpenDrainHigh = true;
masterConfig->baudRate_Hz        = 100000U;
masterConfig->busType            = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with I3C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *i3c_master_config_t*.

void I3C_MasterInit(I3C_Type *base, const *i3c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- masterConfig – User provided peripheral configuration. Use I3C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`void I3C_MasterDeinit(I3C_Type *base)`

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The I3C peripheral base address.

`status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)`

`status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)`

`status_t I3C_CheckForBusyBus(I3C_Type *base)`

`static inline void I3C_MasterEnable(I3C_Type *base, i3c_master_enable_t enable)`

Set I3C module master mode.

Parameters

- `base` – The I3C peripheral base address.
- `enable` – Enable master mode.

`void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)`

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableSlave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

`void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_Hz)`

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

`void I3C_SlaveDeinit(I3C_Type *base)`

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- `base` – The I3C peripheral base address.

static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)
 Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnabled – Enable or disable.

static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)
 Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_master_flags`

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)
 Clears the I3C master status flag state.

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag
- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

Attempts to clear other flags has no effect.

See also:

`_i3c_master_flags`.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)
 Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:`_i3c_master_error_flags`**Parameters**

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master error status flag state.

See also:`_i3c_master_error_flags`.**Parameters**

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_master_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
i3c_master_state_t I3C_MasterGetState(I3C_Type *base)
```

Gets the I3C master state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C master state.

```
static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)
```

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:`_i3c_slave_flags`**Parameters**

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`

- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

```
static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

```
i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)
```

Gets the I3C slave state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)

static inline void I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C master interrupt requests.

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C master interrupt requests.

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_MasterGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag

- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx,  
                                     uint32_t width)
```

Enables or disables I3C master DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

```
static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

```
static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t  
                                     width)
```

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_MasterSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,  
                                           i3c_rx_trigger_level_t rxLvl, bool flushTx, bool  
                                           flushRx)
```

Sets the watermarks for I3C master FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C master FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txCount* – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- *rxCount* – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,  
                                           i3c_rx_trigger_level_t rxLvl, bool flushTx, bool  
                                           flushRx)
```

Sets the watermarks for I3C slave FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.


```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
void I3C_MasterSetBaudRate(I3C_Type *base, const i3c_baudrate_hz_t *baudRate_Hz, uint32_t sourceClock_Hz)
```

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

- `base` – The I3C peripheral base address.
- `baudRate_Hz` – Pointer to structure of requested bus frequency in Hertz.
- `sourceClock_Hz` – I3C functional clock frequency in Hertz.

```
static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The I3C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t I3C_MasterStartWithRxSize(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir, uint8_t rxSize)
```

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the `address` parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

`status_t I3C_MasterStart(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)`

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

`status_t I3C_MasterRepeatedStartWithRxSize(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir, uint8_t rxSize)`

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the *rxSize* = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the *rxSize* configuration.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- *rxSize* – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
static inline status_t I3C_MasterRepeatedStart(I3C_Type *base, i3c_bus_type_t type, uint8_t  
                                              address, i3c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like I3C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.

```
status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
```

Performs a polling send transfer on the I2C/I3C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_I3C_Nak.

Parameters

- *base* – The I3C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.
- *flags* – Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Nak – The slave device sent a NAK in response to an address.
- kStatus_I3C_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_WriteFifoError – Write to M/SWDATAB register when FIFO full.
- kStatus_I3C_InvalidReq – Invalid use of request.

```
status_t I3C_MasterReceive(I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)
```

Performs a polling receive transfer on the I2C/I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.
- `flags` – Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options.

Return values

- `kStatus__Success` – Data was received successfully.
- `kStatus__I3C__Busy` – Another master is currently utilizing the bus.
- `kStatus__I3C__Timeout` – The module has stalled too long in a frame.
- `kStatus__I3C__Term` – The master terminates slave read.
- `kStatus__I3C__HdrParityError` – Parity error from DDR read.
- `kStatus__I3C__CrcError` – CRC error from DDR read.
- `kStatus__I3C__MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus__I3C__ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus__I3C__InvalidReq` – Invalid use of request.

`status_t I3C_MasterStop(I3C_Type *base)`

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The I3C peripheral base address.

Return values

- `kStatus__Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus__I3C__Busy` – Another master is currently utilizing the bus.
- `kStatus__I3C__Timeout` – The module has stalled too long in a frame.
- `kStatus__I3C__InvalidReq` – Invalid use of request.

`void I3C_MasterEmitRequest(I3C_Type *base, i3c_bus_request_t masterReq)`

I3C master emit request.

Parameters

- `base` – The I3C peripheral base address.
- `masterReq` – I3C master request of type `i3c_bus_request_t`

`static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, i3c_ibi_response_t ibiResponse)`

I3C master emit request.

Parameters

- `base` – The I3C peripheral base address.
- `ibiResponse` – I3C master emit IBI response of type `i3c_ibi_response_t`

```
void I3C_MasterRegisterIBI(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)
```

I3C master register IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to ibi rule description of type `i3c_register_ibi_addr_t`

```
void I3C_MasterGetIBIRules(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)
```

I3C master get IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to store the read out ibi rule description.

```
i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)
```

I3C master get IBI Type.

Parameters

- base – The I3C peripheral base address.

Return values

`i3c_ibi_type_t` – Type of `i3c_ibi_type_t`.

```
static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)
```

I3C master get IBI Address.

Parameters

- base – The I3C peripheral base address.

Return values

The – 8-bit IBI address.

```
status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t  
count, i3c_master_daa_baudrate_t  
*daaBaudRate)
```

Performs a DAA in the i3c bus with specified temporary baud rate.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list.
- daaBaudRate – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

```
static inline status_t I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t  
count)
```

Performs a DAA in the i3c bus.

Parameters

- `base` – The I3C peripheral base address.
- `addressList` – The pointer for address list which is used to do DAA.
- `count` – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- `kStatus_I3C_SlaveCountExceed` – The I3C slave count has exceed the definition in `I3C_MAX_DEVCNT`.

`i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)`

Get device information list after DAA process is done.

Parameters

- `base` – The I3C peripheral base address.
- `count` – **[out]** The pointer to store the available device count.

Returns

Pointer to the `i3c_device_info_t` array.

`void I3C_MasterClearDeviceCount(I3C_Type *base)`

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

Parameters

- `base` – The I3C peripheral base address.

`status_t I3C_MasterTransferBlocking(I3C_Type *base, i3c_master_transfer_t *transfer)`

Performs a master polling transfer on the I2C/I3C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The I3C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_IBIWon` – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.

- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`void I3C_MasterTransferCreateHandle(I3C_Type *base, i3c_master_handle_t *handle, const i3c_master_transfer_callback_t *callback, void *userData)`

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C master driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_MasterTransferNonBlocking(I3C_Type *base, i3c_master_handle_t *handle, i3c_master_transfer_t *transfer)`

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus__Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I3C_MasterTransferGetCount(I3C_Type *base, *i3c_master_handle_t* *handle, *size_t* *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus__Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void I3C_MasterTransferAbort(I3C_Type *base, *i3c_master_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferHandleIRQ(I3C_Type *base, *void* *intHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to the I3C master driver handle.

enum _i3c_master_flags

I3C master peripheral flags.

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag

- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterBetweenFlag

Between messages/DAA's flag

enumerator kI3C_MasterNackDetectFlag

NACK detected flag

enumerator kI3C_MasterSlaveStartFlag

Slave request start flag

enumerator kI3C_MasterControlDoneFlag

Master request complete flag

enumerator kI3C_MasterCompleteFlag

Transfer complete flag

enumerator kI3C_MasterRxReadyFlag

Rx data ready in Rx buffer flag

enumerator kI3C_MasterTxReadyFlag

Tx buffer ready for Tx data flag

enumerator kI3C_MasterArbitrationWonFlag

Header address won arbitration flag

enumerator kI3C_MasterErrorFlag

Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag

Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum _i3c_master_error_flags

I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag

Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag

Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag

Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag

CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag

Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag

Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag

Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag

The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags

All error flags

enum _i3c_master_state

I3C working master state.

Values:

enumerator kI3C_MasterStateIdle

Bus stopped.

enumerator kI3C_MasterStateSlvReq

Bus stopped but slave holding SDA low.

enumerator kI3C_MasterStateMsgSdr

In SDR Message mode from using MWMSG_SDR.

enumerator kI3C_MasterStateNormAct

In normal active SDR mode.

enumerator kI3C_MasterStateDdr

In DDR Message mode.

enumerator kI3C_MasterStateDaa

In ENTDAAs mode.

enumerator kI3C_MasterStateIbiAck

Waiting on IBI ACK/NACK decision.

enumerator kI3C_MasterStateIbiRcv

Receiving IBI.

enum _i3c_master_enable

I3C master enable configuration.

Values:

enumerator kI3C_MasterOff

Master off.

enumerator kI3C_MasterOn

Master on.

enumerator kI3C_MasterCapable

Master capable.

enum _i3c_master_hkeep

I3C high keeper configuration.

Values:

enumerator kI3C_MasterHighKeeperNone

Use PUR to hold SCL high.

enumerator kI3C_MasterHighKeeperWiredIn

Use pin_HK controls.

enumerator kI3C_MasterPassiveSDA

Hi-Z for Bus Free and hold SDA.

enumerator kI3C_MasterPassiveSDASCL

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum _i3c_bus_request

Emits the requested operation when doing in pieces vs. by message.

Values:

enumerator kI3C_RequestNone

No request.

enumerator kI3C_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA

Process DAA.

enumerator kI3C_RequestForceExit

Request to force exit.

enumerator kI3C_RequestAutoIbi

Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type

Bus type with EmitStartAddr.

Values:

enumerator kI3C_TypeI3CSdr

SDR mode of I3C.

enumerator kI3C_TypeI2C

Standard i2c protocol.

enumerator kI3C_TypeI3CDdr

HDR-DDR mode of I3C.

enum _i3c_ibi_response

IBI response.

Values:

enumerator kI3C_IbiRespAck

ACK with no mandatory byte.

enumerator kI3C_IbiRespNack
NACK.

enumerator kI3C_IbiRespAckMandatory
ACK with mandatory byte.

enumerator kI3C_IbiRespManual
Reserved.

enum _i3c_ibi_type
IBI type.

Values:

enumerator kI3C_IbiNormal
In-band interrupt.

enumerator kI3C_IbiHotJoin
slave hot join.

enumerator kI3C_IbiMasterRequest
slave master ship request.

enum _i3c_ibi_state
IBI state.

Values:

enumerator kI3C_IbiReady
In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed
In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending
In-band interrupt Ack/Nack pending for decision.

enum _i3c_direction
Direction of master and slave transfers.

Values:

enumerator kI3C_Write
Master transmit.

enumerator kI3C_Read
Master receive.

enum _i3c_tx_trigger_level
Watermark of TX int/dma trigger level.

Values:

enumerator kI3C_TxTriggerOnEmpty
Trigger on empty.

enumerator kI3C_TxTriggerUntilOneQuarterOrLess
Trigger on 1/4 full or less.

enumerator kI3C_TxTriggerUntilOneHalfOrLess
Trigger on 1/2 full or less.

enumerator kI3C_TxTriggerUntilOneLessThanFull
Trigger on 1 less than full or less.

enum _i3c_rx_trigger_level

Watermark of RX int/dma trigger level.

Values:

enumerator kI3C_RxTriggerOnNotEmpty

Trigger on not empty.

enumerator kI3C_RxTriggerUntilOneQuarterOrMore

Trigger on 1/4 full or more.

enumerator kI3C_RxTriggerUntilOneHalfOrMore

Trigger on 1/2 full or more.

enumerator kI3C_RxTriggerUntilThreeQuarterOrMore

Trigger on 3/4 full or more.

enum _i3c_rx_term_ops

I3C master read termination operations.

Values:

enumerator kI3C_RxTermDisable

Master doesn't terminate read, used for CCC transfer.

enumerator kI3C_RxAutoTerm

Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C_RxTermLastByte

Master terminates read at any time after START, no length limitation.

enum _i3c_start_scl_delay

I3C start SCL delay options.

Values:

enumerator kI3C_NoDelay

No delay.

enumerator kI3C_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum _i3c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_i3c_master_transfer::flags` field.

Values:

enumerator kI3C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator `kI3C_TransferRepeatedStartFlag`
Send a repeated start condition

enumerator `kI3C_TransferNoStopFlag`
Don't send a stop condition.

enumerator `kI3C_TransferWordsFlag`
Transfer in words, else transfer in bytes.

enumerator `kI3C_TransferDisableRxTermFlag`
Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator `kI3C_TransferRxAutoTermFlag`
Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in `I3C_MasterReceive`.

enumerator `kI3C_TransferStartWithBroadcastAddr`
Start transfer with 0x7E, then read/write data with device address.

typedef enum `_i3c_master_state` `i3c_master_state_t`
I3C working master state.

typedef enum `_i3c_master_enable` `i3c_master_enable_t`
I3C master enable configuration.

typedef enum `_i3c_master_hkeep` `i3c_master_hkeep_t`
I3C high keeper configuration.

typedef enum `_i3c_bus_request` `i3c_bus_request_t`
Emits the requested operation when doing in pieces vs. by message.

typedef enum `_i3c_bus_type` `i3c_bus_type_t`
Bus type with `EmitStartAddr`.

typedef enum `_i3c_ibi_response` `i3c_ibi_response_t`
IBI response.

typedef enum `_i3c_ibi_type` `i3c_ibi_type_t`
IBI type.

typedef enum `_i3c_ibi_state` `i3c_ibi_state_t`
IBI state.

typedef enum `_i3c_direction` `i3c_direction_t`
Direction of master and slave transfers.

typedef enum `_i3c_tx_trigger_level` `i3c_tx_trigger_level_t`
Watermark of TX int/dma trigger level.

typedef enum `_i3c_rx_trigger_level` `i3c_rx_trigger_level_t`
Watermark of RX int/dma trigger level.

typedef enum `_i3c_rx_term_ops` `i3c_rx_term_ops_t`
I3C master read termination operations.

typedef enum `_i3c_start_scl_delay` `i3c_start_scl_delay_t`
I3C start SCL delay options.

typedef struct `_i3c_register_ibi_addr` `i3c_register_ibi_addr_t`
Structure with setting master IBI rules and slave registry.

typedef struct `_i3c_baudrate` `i3c_baudrate_hz_t`
Structure with I3C baudrate settings.

```
typedef struct _i3c_master_daa_baudrate i3c_master_daa_baudrate_t
```

I3C DAA baud rate configuration.

```
typedef struct _i3c_master_config i3c_master_config_t
```

Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef struct _i3c_master_transfer i3c_master_transfer_t
```

```
typedef struct _i3c_master_handle i3c_master_handle_t
```

```
typedef struct _i3c_master_transfer_callback i3c_master_transfer_callback_t
```

i3c master callback functions.

```
typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
```

Typedef for master interrupt handler.

```
struct _i3c_register_ibi_addr
```

#include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.

Public Members

```
uint8_t address[5]
```

Address array for registry.

```
bool i3cFastStart
```

Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.

```
bool ibiHasPayload
```

Whether the address array has mandatory IBI byte.

```
struct _i3c_baudrate
```

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

```
uint32_t i2cBaud
```

Desired I2C baud rate in Hertz.

```
uint32_t i3cPushPullBaud
```

Desired I3C push-pull baud rate in Hertz.

```
uint32_t i3cOpenDrainBaud
```

Desired I3C open-drain baud rate in Hertz.

```
struct _i3c_master_daa_baudrate
```

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

```
uint32_t sourceClock_Hz
```

FCLK, function clock in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct __i3c_master_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

struct __i3c_master_transfer_callback

#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)

Transfer complete callback

void (*ibiCallback)(I3C_Type *base, i3c_master_handle_t *handle, i3c_ibi_type_t ibiType, i3c_ibi_state_t ibiState)

IBI event callback

void (*transferComplete)(I3C_Type *base, i3c_master_handle_t *handle, status_t completionStatus, void *userData)

Transfer complete callback

struct __i3c_master_transfer

#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options. Set to 0 or `kI3C_TransferDefaultFlag` for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i3c_direction_t direction

Either `kI3C_Read` or `kI3C_Write`.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

i3c_bus_type_t busType

bus type.

i3c_ibi_response_t ibiResponse

ibi response during transfer.

struct `_i3c_master_handle`

#include <fsl_i3c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint32_t remainingBytes

Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps

Read termination operation.

i3c_master_transfer_t transfer

Copy of the current transfer info.

uint8_t ibiAddress

Slave address which request IBI.

uint8_t *ibiBuff

Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize

IBI payload size.

i3c_ibi_type_t ibiType

IBI type.

i3c_master_transfer_callback_t callback

Callback functions pointer.

void *userData

Application data passed to callback.

2.47 I3C Slave DMA Driver

```
void I3C_SlaveTransferCreateHandleDMA(I3C_Type *base, i3c_slave_dma_handle_t *handle,  
                                     i3c_slave_dma_callback_t callback, void *userData,  
                                     dma_handle_t *rxDmaHandle, dma_handle_t  
                                     *txDmaHandle)
```

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.
- rxDmaHandle – Handle for the DMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_SlaveTransferDMA(I3C_Type *base, i3c_slave_dma_handle_t *handle,  
                             i3c_slave_dma_transfer_t *transfer, uint32_t eventMask)
```

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.
- transfer – The pointer to the transfer descriptor.
- eventMask – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
void I3C_SlaveTransferAbortDMA(I3C_Type *base, i3c_slave_dma_handle_t *handle)
```

Abort a slave dma non-blocking transfer in a early time.

Parameters

- base – I3C peripheral base address
- handle – pointer to i3c_slave_dma_handle_t structure

```
void I3C_SlaveTransferDMAHandleIRQ(I3C_Type *base, void *i3CHandle)
```

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave DMA driver handle.

```
typedef struct _i3c_slave_dma_handle i3c_slave_dma_handle_t
```

```
typedef struct _i3c_slave_dma_transfer i3c_slave_dma_transfer_t
```

I3C slave transfer structure.

```
typedef void (*i3c_slave_dma_callback_t)(I3C_Type *base, i3c_slave_dma_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

Param base

Base address for the I3C instance on which the event occurred.

Param handle

Pointer to slave DMA transfer handle.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the call-back.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _i3c_slave_dma_transfer
```

```
#include <fsl_i3c_dma.h> I3C slave transfer structure.
```

```
struct _i3c_slave_dma_handle
```

```
#include <fsl_i3c_dma.h> I3C slave dma handle structure.
```

Note: The contents of this structure are private and subject to change.

2.48 I3C Slave Driver

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

```
void I3C_SlaveDeinit(I3C_Type *base)
```

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- `base` – The I3C peripheral base address.

```
static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)
    Enable/Disable Slave.
```

Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

```
static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)
```

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

```
static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)

Gets the I3C slave state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C slave activity state, refer *i3c_slave_activity_state_t*.

status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl,  
                                         i3c_rx_trigger_level_t rxLvl, bool flushTx, bool  
                                         flushRx)
```

Sets the watermarks for I3C slave FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- *flushTx* – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- *flushRx* – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

```
static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txCount* – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- *rxCount* – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

```
status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)
```

Performs a polling send transfer on the I3C bus.

Parameters

- *base* – The I3C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

```
status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)
```

Performs a polling receive transfer on the I3C bus.

Parameters

- *base* – The I3C peripheral base address.
- *rxBuff* – The pointer to the data to be transferred.
- *rxSize* – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

```
void I3C_SlaveTransferCreateHandle(I3C_Type *base, i3c_slave_handle_t *handle,  
                                  i3c_slave_transfer_callback_t callback, void *userData)
```

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i3c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI3C_SlaveTransmitEvent` and `kI3C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI3C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI3C_SlaveAllEvents` to enable all events.

Return values

- `kStatus_Success` – Slave transfers were successfully started.
- `kStatus_I3C_Busy` – Slave transfers have already been started on this handle.

`status_t I3C_SlaveTransferGetCount(I3C_Type *base, i3c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` –

```
void I3C_SlaveTransferAbort(I3C_Type *base, i3c_slave_handle_t *handle)
```

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

```
void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)
```

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

```
enum _i3c_slave_flags
```

I3C slave peripheral flags.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveNotStopFlag
Slave status not stop flag

enumerator kI3C_SlaveMessageFlag
Slave status message, indicating slave is listening to the bus traffic or responding

enumerator kI3C_SlaveRequiredReadFlag
Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator kI3C_SlaveRequiredWriteFlag
Slave status request write, master is doing SDR write to slave, except slave in ENTDAAMode

enumerator kI3C_SlaveBusDAAModeFlag
I3C bus is in ENTDAAMode

enumerator kI3C_SlaveBusHDRModeFlag
I3C bus is in HDR mode

enumerator kI3C_SlaveBusStartFlag
Start/Re-start event is seen since the bus was last cleared

enumerator kI3C_SlaveMatchedFlag
Slave address(dynamic/static) matched since last cleared

enumerator kI3C_SlaveBusStopFlag
Stop event is seen since the bus was last cleared

enumerator kI3C_SlaveRxReadyFlag
Rx data ready in rx buffer flag

enumerator kI3C_SlaveTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_SlaveDynamicAddrChangedFlag
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C_SlaveReceivedCCCFlag
Slave received Common command code

enumerator kI3C_SlaveErrorFlag
Error occurred flag

enumerator kI3C_SlaveHDRCommandMatchFlag
High data rate command match

enumerator kI3C_SlaveCCCHandledFlag
Slave received Common command code is handled by I3C module

enumerator kI3C_SlaveEventSentFlag
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C_SlaveIbiDisableFlag
Slave in band interrupt is disabled.

enumerator kI3C_SlaveMasterRequestDisabledFlag
Slave master request is disabled.

enumerator kI3C_SlaveHotJoinDisabledFlag
Slave Hot-Join is disabled.

enumerator kI3C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C_SlaveAllIrqFlags

enum _i3c_slave_error_flags

I3C slave error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveErrorOverrunFlag

Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C_SlaveErrorUnderrunFlag

Slave internal to-bus buffer/FIFO underrun

enumerator kI3C_SlaveErrorUnderrunNakFlag

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C_SlaveErrorTermFlag

Terminate error from master

enumerator kI3C_SlaveErrorInvalidStartFlag

Slave invalid start flag

enumerator kI3C_SlaveErrorSdrParityFlag

SDR parity error

enumerator kI3C_SlaveErrorHdrParityFlag

HDR parity error

enumerator kI3C_SlaveErrorHdrCRCFlag

HDR-DDR CRC error

enumerator kI3C_SlaveErrorS0S1Flag

S0 or S1 error

enumerator kI3C_SlaveErrorOverreadFlag

Over-read error

enumerator kI3C_SlaveErrorOverwriteFlag

Over-write error

enum _i3c_slave_event

I3C slave.event.

Values:

enumerator kI3C_SlaveEventNormal

Normal mode.

enumerator kI3C_SlaveEventIBI

In band interrupt event.

enumerator kI3C_SlaveEventMasterReq

Master request event.

enumerator kI3C_SlaveEventHotJoinReq

Hot-join event.

enum _i3c_slave_activity_state

I3C slave.activity state.

Values:

enumerator kI3C_SlaveNoLatency

Normal bus operation

enumerator kI3C_SlaveLatency1Ms

1ms of latency.

enumerator kI3C_SlaveLatency100Ms

100ms of latency.

enumerator kI3C_SlaveLatency10S

10s latency.

enum _i3c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I3C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kI3C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kI3C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI3C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveRequiredTransmitEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveStartEvent

A start/repeated start was detected.

enumerator kI3C_SlaveHDRCommandMatchEvent

Slave Match HDR Command.

enumerator kI3C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kI3C_SlaveRequestSentEvent

Slave request event sent.

enumerator kI3C_SlaveReceivedCCCEvent

Slave received CCC event, need to handle by application.

enumerator kI3C_SlaveAllEvents

Bit mask of all available events.

typedef enum _i3c_slave_event i3c_slave_event_t

I3C slave.event.

typedef enum _i3c_slave_activity_state i3c_slave_activity_state_t

I3C slave.activity state.

```
typedef struct _i3c_slave_config i3c_slave_config_t
```

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _i3c_slave_transfer_event i3c_slave_transfer_event_t
```

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I3C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

```
typedef struct _i3c_slave_transfer i3c_slave_transfer_t
```

I3C slave transfer structure.

```
typedef struct _i3c_slave_handle i3c_slave_handle_t
```

```
typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `I3C_SlaveSetCallback()` function after you have created a handle.

Param base

Base address for the I3C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
```

Typedef for slave interrupt handler.

```
struct _i3c_slave_config
```

#include <fsl_i3c.h> Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

`uint16_t` vendorID
Device vendor ID(manufacture ID).

`uint32_t` partNumber
Device part number info

`uint8_t` dcr
Device characteristics register information.

`uint8_t` bcr
Bus characteristics register information.

`uint8_t` hdrMode
Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.

`bool` nakAllRequest
Whether to reply NAK to all requests except broadcast CCC.

`bool` ignoreS0S1Error
Whether to ignore S0/S1 error in SDR mode.

`bool` offline
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

`bool` matchSlaveStartStop
Whether to assert start/stop status only the time slave is addressed.

`uint32_t` maxWriteLength
Maximum write length.

`uint32_t` maxReadLength
Maximum read length.

`struct _i3c_slave_transfer`
#include <fsl_i3c.h> I3C slave transfer structure.

Public Members

`uint32_t` event
Reason the callback is being invoked.

`uint8_t *txData`
Transfer buffer

`size_t` txDataSize
Transfer size

`uint8_t *rxData`
Transfer buffer

`size_t` rxDataSize
Transfer size

`status_t` completionStatus
Success or error code describing how the transfer completed. Only applies for `kI3C_SlaveCompletionEvent`.

`size_t` transferredCount
Number of bytes actually transferred since start or last repeated start.

struct `_i3c_slave_handle`
#include <fsl_i3c.h> I3C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

i3c_slave_transfer_t transfer
I3C slave transfer copy.

bool isBusy
Whether transfer is busy.

bool wasTransmit
Whether the last transfer was a transmit.

uint32_t eventMask
Mask of enabled events.

uint32_t transferredCount
Count of bytes transferred.

i3c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

size_t txFifoSize
Tx Fifo size

2.49 INPUTMUX: Input Multiplexing Driver

enum `_inputmux_connection_t`
INPUTMUX connections type.

Values:

enumerator `kINPUTMUX_SctGpioInAToSct0`
SCT0 INMUX.

enumerator `kINPUTMUX_SctGpioInBToSct0`

enumerator `kINPUTMUX_SctGpioInCToSct0`

enumerator `kINPUTMUX_SctGpioInDToSct0`

enumerator `kINPUTMUX_SctGpioInEToSct0`

enumerator `kINPUTMUX_SctGpioInFToSct0`

enumerator `kINPUTMUX_SctGpioInGToSct0`

enumerator `kINPUTMUX_SctGpioInHToSct0`

enumerator `kINPUTMUX_Ctimer0M0ToSct0`

enumerator kINPUTMUX__Ctimer1M0ToSct0
enumerator kINPUTMUX__Ctimer2M0ToSct0
enumerator kINPUTMUX__Ctimer3M0ToSct0
enumerator kINPUTMUX__Ctimer4M0ToSct0
enumerator kINPUTMUX__Adc0IrqToSct0
enumerator kINPUTMUX__GpioIntBmatchToSct0
enumerator kINPUTMUX__Usb0FrameToggleToSct0
enumerator kINPUTMUX__CompOutToSct0
enumerator kINPUTMUX__I2sSharedSck0ToSct0
enumerator kINPUTMUX__I2sSharedSck1ToSct0
enumerator kINPUTMUX__I2sSharedWs0ToSct0
enumerator kINPUTMUX__I2sSharedWs1ToSct0
enumerator kINPUTMUX__ArmTxeVToSct0
enumerator kINPUTMUX__DebugHaltedToSct0
enumerator kINPUTMUX__Adc1IrqToSct0
enumerator kINPUTMUX__Adc0tcomp0ToSct0
enumerator kINPUTMUX__Adc0tcomp1ToSct0
enumerator kINPUTMUX__Adc0tcomp2ToSct0
enumerator kINPUTMUX__Adc0tcomp3ToSct0
enumerator kINPUTMUX__Adc1tcomp0ToSct0
enumerator kINPUTMUX__Adc1tcomp1ToSct0
enumerator kINPUTMUX__Adc1tcomp2ToSct0
enumerator kINPUTMUX__Adc1tcomp3ToSct0
enumerator kINPUTMUX__Hscmp0OutToSct0
enumerator kINPUTMUX__Hscmp1OutToSct0
enumerator kINPUTMUX__Hscmp2OutToSct0
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToSct0
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToSct0
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToSct0
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToSct0
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToSct0
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToSct0
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToSct0

enumerator kINPUTMUX_Pwm1Sm3MuxTrig01ToSct0

enumerator kINPUTMUX_Enc0CmpPosMatchToSct0

enumerator kINPUTMUX_Enc1CmpPosMatchToSct0

enumerator kINPUTMUX_Aoi0Out0ToSct0

enumerator kINPUTMUX_Aoi0Out1ToSct0

enumerator kINPUTMUX_Aoi0Out2ToSct0

enumerator kINPUTMUX_Aoi0Out3ToSct0

enumerator kINPUTMUX_Aoi1Out0ToSct0

enumerator kINPUTMUX_Aoi1Out1ToSct0

enumerator kINPUTMUX_Aoi1Out2ToSct0

enumerator kINPUTMUX_Aoi1Out3ToSct0

enumerator kINPUTMUX_Fc3SckToSct0

enumerator kINPUTMUX_Fc3RxdSdaMosiDataToSct0

enumerator kINPUTMUX_Fc3TxdSclMisoWsToSct0

enumerator kINPUTMUX_Fc3CtsDsaSsel0ToSct0

enumerator kINPUTMUX_TmprOutToSct0

TIMER0 CAPTSEL.

enumerator kINPUTMUX_CtimerInp0ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp1ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp2ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp3ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp4ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp5ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp6ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp7ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp8ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp9ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp10ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp11ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp12ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp13ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp14ToTimer0Captsel

enumerator kINPUTMUX_CtimerInp15ToTimer0Captsel

enumerator kINPUTMUX__CtimerInp16ToTimer0Captsel
enumerator kINPUTMUX__CtimerInp17ToTimer0Captsel
enumerator kINPUTMUX__CtimerInp18ToTimer0Captsel
enumerator kINPUTMUX__CtimerInp19ToTimer0Captsel
enumerator kINPUTMUX__Usb0FrameToggleToTimer0Captsel
enumerator kINPUTMUX__CompOutToTimer0Captsel
enumerator kINPUTMUX__I2sSharedWs0ToTimer0Captsel
enumerator kINPUTMUX__I2sSharedWs1ToTimer0Captsel
enumerator kINPUTMUX__Adc0IrqToTimer0Captsel
enumerator kINPUTMUX__Adc1IrqToTimer0Captsel
enumerator kINPUTMUX__Hscmp0OutToTimer0Captsel
enumerator kINPUTMUX__Hscmp1OutToTimer0Captsel
enumerator kINPUTMUX__Hscmp2OutToTimer0Captsel
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToTimer0Captsel
enumerator kINPUTMUX__Enc0CmpPosMatchToTimer0Captsel
enumerator kINPUTMUX__Enc1CmpPosMatchToTimer0Captsel
enumerator kINPUTMUX__Aoi0Out0ToTimer0Captsel
enumerator kINPUTMUX__Aoi0Out1ToTimer0Captsel
enumerator kINPUTMUX__Aoi0Out2ToTimer0Captsel
enumerator kINPUTMUX__Aoi0Out3ToTimer0Captsel
enumerator kINPUTMUX__Aoi1Out0ToTimer0Captsel
enumerator kINPUTMUX__Aoi1Out1ToTimer0Captsel
enumerator kINPUTMUX__Aoi1Out2ToTimer0Captsel
enumerator kINPUTMUX__Aoi1Out3ToTimer0Captsel
enumerator kINPUTMUX__TmprOutToTimer0Captsel
TIMERO Trigger.

enumerator kINPUTMUX_CtimerInp0ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp1ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp2ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp3ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp4ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp5ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp6ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp7ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp8ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp9ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp10ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp11ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp12ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp13ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp14ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp15ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp16ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp17ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp18ToTimer0Trigger
enumerator kINPUTMUX_CtimerInp19ToTimer0Trigger
enumerator kINPUTMUX_Usb0FrameToggleToTimer0Trigger
enumerator kINPUTMUX_CompOutToTimer0Trigger
enumerator kINPUTMUX_I2sSharedWs0ToTimer0Trigger
enumerator kINPUTMUX_I2sSharedWs1ToTimer0Trigger
enumerator kINPUTMUX_Adc0IrqToTimer0Trigger
enumerator kINPUTMUX_Adc1IrqToTimer0Trigger
enumerator kINPUTMUX_Hscmp0OutToTimer0Trigger
enumerator kINPUTMUX_Hscmp1OutToTimer0Trigger
enumerator kINPUTMUX_Hscmp2OutToTimer0Trigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm1Sm0MuxTrig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig01ToTimer0Trigger
enumerator kINPUTMUX_Enc0CmpPosMatchToTimer0Trigger
enumerator kINPUTMUX_Enc1CmpPosMatchToTimer0Trigger
enumerator kINPUTMUX_Aoi0Out0ToTimer0Trigger
enumerator kINPUTMUX_Aoi0Out1ToTimer0Trigger
enumerator kINPUTMUX_Aoi0Out2ToTimer0Trigger
enumerator kINPUTMUX_Aoi0Out3ToTimer0Trigger
enumerator kINPUTMUX_Aoi1Out0ToTimer0Trigger
enumerator kINPUTMUX_Aoi1Out1ToTimer0Trigger
enumerator kINPUTMUX_Aoi1Out2ToTimer0Trigger
enumerator kINPUTMUX_Aoi1Out3ToTimer0Trigger
enumerator kINPUTMUX_TmprOutToTimer0Trigger

TIMER1 CAPTSEL.

enumerator kINPUTMUX_CtimerInp0ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp1ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp2ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp3ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp4ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp5ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp6ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp7ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp8ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp9ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp10ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp11ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp12ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp13ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp14ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp15ToTimer1Captsel
enumerator kINPUTMUX_CtimerInp16ToTimer1Captsel

enumerator kINPUTMUX__CtimerInp17ToTimer1Captsel
enumerator kINPUTMUX__CtimerInp18ToTimer1Captsel
enumerator kINPUTMUX__CtimerInp19ToTimer1Captsel
enumerator kINPUTMUX__Usb0FrameToggleToTimer1Captsel
enumerator kINPUTMUX__CompOutToTimer1Captsel
enumerator kINPUTMUX__I2sSharedWs0ToTimer1Captsel
enumerator kINPUTMUX__I2sSharedWs1ToTimer1Captsel
enumerator kINPUTMUX__Adc0IrqToTimer1Captsel
enumerator kINPUTMUX__Adc1IrqToTimer1Captsel
enumerator kINPUTMUX__Hscmp0OutToTimer1Captsel
enumerator kINPUTMUX__Hscmp1OutToTimer1Captsel
enumerator kINPUTMUX__Hscmp2OutToTimer1Captsel
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToTimer1Captsel
enumerator kINPUTMUX__Enc0CmpPosMatchToTimer1Captsel
enumerator kINPUTMUX__Enc1CmpPosMatchToTimer1Captsel
enumerator kINPUTMUX__Aoi0Out0ToTimer1Captsel
enumerator kINPUTMUX__Aoi0Out1ToTimer1Captsel
enumerator kINPUTMUX__Aoi0Out2ToTimer1Captsel
enumerator kINPUTMUX__Aoi0Out3ToTimer1Captsel
enumerator kINPUTMUX__Aoi1Out0ToTimer1Captsel
enumerator kINPUTMUX__Aoi1Out1ToTimer1Captsel
enumerator kINPUTMUX__Aoi1Out2ToTimer1Captsel
enumerator kINPUTMUX__Aoi1Out3ToTimer1Captsel
enumerator kINPUTMUX__TmprOutToTimer1Captsel
 TIMER1 Trigger.
enumerator kINPUTMUX__CtimerInp0ToTimer1Trigger

enumerator kINPUTMUX_CtimerInp1ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp2ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp3ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp4ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp5ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp6ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp7ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp8ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp9ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp10ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp11ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp12ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp13ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp14ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp15ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp16ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp17ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp18ToTimer1Trigger
enumerator kINPUTMUX_CtimerInp19ToTimer1Trigger
enumerator kINPUTMUX_Usb0FrameToggleToTimer1Trigger
enumerator kINPUTMUX_CompOutToTimer1Trigger
enumerator kINPUTMUX_I2sSharedWs0ToTimer1Trigger
enumerator kINPUTMUX_I2sSharedWs1ToTimer1Trigger
enumerator kINPUTMUX_Adc0IrqToTimer1Trigger
enumerator kINPUTMUX_Adc1IrqToTimer1Trigger
enumerator kINPUTMUX_Hscmp0OutToTimer1Trigger
enumerator kINPUTMUX_Hscmp1OutToTimer1Trigger
enumerator kINPUTMUX_Hscmp2OutToTimer1Trigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm1Sm1MuxTrig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig01ToTimer1Trigger
enumerator kINPUTMUX_Enc0CmpPosMatchToTimer1Trigger
enumerator kINPUTMUX_Enc1CmpPosMatchToTimer1Trigger
enumerator kINPUTMUX_Aoi0Out0ToTimer1Trigger
enumerator kINPUTMUX_Aoi0Out1ToTimer1Trigger
enumerator kINPUTMUX_Aoi0Out2ToTimer1Trigger
enumerator kINPUTMUX_Aoi0Out3ToTimer1Trigger
enumerator kINPUTMUX_Aoi1Out0ToTimer1Trigger
enumerator kINPUTMUX_Aoi1Out1ToTimer1Trigger
enumerator kINPUTMUX_Aoi1Out2ToTimer1Trigger
enumerator kINPUTMUX_Aoi1Out3ToTimer1Trigger
enumerator kINPUTMUX_TmprOutToTimer1Trigger
TIMER2 CAPTSEL.
enumerator kINPUTMUX_CtimerInp0ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp1ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp2ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp3ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp4ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp5ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp6ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp7ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp8ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp9ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp10ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp11ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp12ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp13ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp14ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp15ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp16ToTimer2Cptsel
enumerator kINPUTMUX_CtimerInp17ToTimer2Cptsel

enumerator kINPUTMUX__CtimerInp18ToTimer2Cptsel
enumerator kINPUTMUX__CtimerInp19ToTimer2Cptsel
enumerator kINPUTMUX__Usb0FrameToggleToTimer2Cptsel
enumerator kINPUTMUX__CompOutToTimer2Cptsel
enumerator kINPUTMUX__I2sSharedWs0ToTimer2Cptsel
enumerator kINPUTMUX__I2sSharedWs1ToTimer2Cptsel
enumerator kINPUTMUX__Adc0IrqToTimer2Cptsel
enumerator kINPUTMUX__Adc1IrqToTimer2Cptsel
enumerator kINPUTMUX__Hscmp0OutToTimer2Cptsel
enumerator kINPUTMUX__Hscmp1OutToTimer2Cptsel
enumerator kINPUTMUX__Hscmp2OutToTimer2Cptsel
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToTimer2Cptsel
enumerator kINPUTMUX__Enc0CmpPosMatchToTimer2Cptsel
enumerator kINPUTMUX__Enc1CmpPosMatchToTimer2Cptsel
enumerator kINPUTMUX__Aoi0Out0ToTimer2Cptsel
enumerator kINPUTMUX__Aoi0Out1ToTimer2Cptsel
enumerator kINPUTMUX__Aoi0Out2ToTimer2Cptsel
enumerator kINPUTMUX__Aoi0Out3ToTimer2Cptsel
enumerator kINPUTMUX__Aoi1Out0ToTimer2Cptsel
enumerator kINPUTMUX__Aoi1Out1ToTimer2Cptsel
enumerator kINPUTMUX__Aoi1Out2ToTimer2Cptsel
enumerator kINPUTMUX__Aoi1Out3ToTimer2Cptsel
enumerator kINPUTMUX__TmprOutToTimer2Cptsel
TIMER2 Trigger.
enumerator kINPUTMUX__CtimerInp0ToTimer2Trigger
enumerator kINPUTMUX__CtimerInp1ToTimer2Trigger

enumerator kINPUTMUX_CtimerInp2ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp3ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp4ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp5ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp6ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp7ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp8ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp9ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp10ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp11ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp12ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp13ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp14ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp15ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp16ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp17ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp18ToTimer2Trigger
enumerator kINPUTMUX_CtimerInp19ToTimer2Trigger
enumerator kINPUTMUX_Usb0FrameToggleToTimer2Trigger
enumerator kINPUTMUX_CompOutToTimer2Trigger
enumerator kINPUTMUX_I2sSharedWs0ToTimer2Trigger
enumerator kINPUTMUX_I2sSharedWs1ToTimer2Trigger
enumerator kINPUTMUX_Adc0IrqToTimer2Trigger
enumerator kINPUTMUX_Adc1IrqToTimer2Trigger
enumerator kINPUTMUX_Hscmp0OutToTimer2Trigger
enumerator kINPUTMUX_Hscmp1OutToTimer2Trigger
enumerator kINPUTMUX_Hscmp2OutToTimer2Trigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm1Sm2MuxTrig01ToTimer2Trigger

enumerator kINPUTMUX_Pwm1Sm3MuxTrig01ToTimer2Trigger

enumerator kINPUTMUX_Enc0CmpPosMatchToTimer2Trigger

enumerator kINPUTMUX_Enc1CmpPosMatchToTimer2Trigger

enumerator kINPUTMUX_Aoi0Out0ToTimer2Trigger

enumerator kINPUTMUX_Aoi0Out1ToTimer2Trigger

enumerator kINPUTMUX_Aoi0Out2ToTimer2Trigger

enumerator kINPUTMUX_Aoi0Out3ToTimer2Trigger

enumerator kINPUTMUX_Aoi1Out0ToTimer2Trigger

enumerator kINPUTMUX_Aoi1Out1ToTimer2Trigger

enumerator kINPUTMUX_Aoi1Out2ToTimer2Trigger

enumerator kINPUTMUX_Aoi1Out3ToTimer2Trigger

enumerator kINPUTMUX_TmprOutToTimer2Trigger

Pin interrupt select.

enumerator kINPUTMUX_GpioPort0Pin0ToPintsel

enumerator kINPUTMUX_GpioPort0Pin1ToPintsel

enumerator kINPUTMUX_GpioPort0Pin2ToPintsel

enumerator kINPUTMUX_GpioPort0Pin3ToPintsel

enumerator kINPUTMUX_GpioPort0Pin4ToPintsel

enumerator kINPUTMUX_GpioPort0Pin5ToPintsel

enumerator kINPUTMUX_GpioPort0Pin6ToPintsel

enumerator kINPUTMUX_GpioPort0Pin7ToPintsel

enumerator kINPUTMUX_GpioPort0Pin8ToPintsel

enumerator kINPUTMUX_GpioPort0Pin9ToPintsel

enumerator kINPUTMUX_GpioPort0Pin10ToPintsel

enumerator kINPUTMUX_GpioPort0Pin11ToPintsel

enumerator kINPUTMUX_GpioPort0Pin12ToPintsel

enumerator kINPUTMUX_GpioPort0Pin13ToPintsel

enumerator kINPUTMUX_GpioPort0Pin14ToPintsel

enumerator kINPUTMUX_GpioPort0Pin15ToPintsel

enumerator kINPUTMUX_GpioPort0Pin16ToPintsel

enumerator kINPUTMUX_GpioPort0Pin17ToPintsel

enumerator kINPUTMUX_GpioPort0Pin18ToPintsel

enumerator kINPUTMUX_GpioPort0Pin19ToPintsel
enumerator kINPUTMUX_GpioPort0Pin20ToPintsel
enumerator kINPUTMUX_GpioPort0Pin21ToPintsel
enumerator kINPUTMUX_GpioPort0Pin22ToPintsel
enumerator kINPUTMUX_GpioPort0Pin23ToPintsel
enumerator kINPUTMUX_GpioPort0Pin24ToPintsel
enumerator kINPUTMUX_GpioPort0Pin25ToPintsel
enumerator kINPUTMUX_GpioPort0Pin26ToPintsel
enumerator kINPUTMUX_GpioPort0Pin27ToPintsel
enumerator kINPUTMUX_GpioPort0Pin28ToPintsel
enumerator kINPUTMUX_GpioPort0Pin29ToPintsel
enumerator kINPUTMUX_GpioPort0Pin30ToPintsel
enumerator kINPUTMUX_GpioPort0Pin31ToPintsel
enumerator kINPUTMUX_GpioPort1Pin0ToPintsel
enumerator kINPUTMUX_GpioPort1Pin1ToPintsel
enumerator kINPUTMUX_GpioPort1Pin2ToPintsel
enumerator kINPUTMUX_GpioPort1Pin3ToPintsel
enumerator kINPUTMUX_GpioPort1Pin4ToPintsel
enumerator kINPUTMUX_GpioPort1Pin5ToPintsel
enumerator kINPUTMUX_GpioPort1Pin6ToPintsel
enumerator kINPUTMUX_GpioPort1Pin7ToPintsel
enumerator kINPUTMUX_GpioPort1Pin8ToPintsel
enumerator kINPUTMUX_GpioPort1Pin9ToPintsel
enumerator kINPUTMUX_GpioPort1Pin10ToPintsel
enumerator kINPUTMUX_GpioPort1Pin11ToPintsel
enumerator kINPUTMUX_GpioPort1Pin12ToPintsel
enumerator kINPUTMUX_GpioPort1Pin13ToPintsel
enumerator kINPUTMUX_GpioPort1Pin14ToPintsel
enumerator kINPUTMUX_GpioPort1Pin15ToPintsel
enumerator kINPUTMUX_GpioPort1Pin16ToPintsel
enumerator kINPUTMUX_GpioPort1Pin17ToPintsel
enumerator kINPUTMUX_GpioPort1Pin18ToPintsel
enumerator kINPUTMUX_GpioPort1Pin19ToPintsel

enumerator kINPUTMUX_GpioPort1Pin20ToPintsel
enumerator kINPUTMUX_GpioPort1Pin21ToPintsel
enumerator kINPUTMUX_GpioPort1Pin22ToPintsel
enumerator kINPUTMUX_GpioPort1Pin23ToPintsel
enumerator kINPUTMUX_GpioPort1Pin24ToPintsel
enumerator kINPUTMUX_GpioPort1Pin25ToPintsel
enumerator kINPUTMUX_GpioPort1Pin26ToPintsel
enumerator kINPUTMUX_GpioPort1Pin27ToPintsel
enumerator kINPUTMUX_GpioPort1Pin28ToPintsel
enumerator kINPUTMUX_GpioPort1Pin29ToPintsel
enumerator kINPUTMUX_GpioPort1Pin30ToPintsel
enumerator kINPUTMUX_GpioPort1Pin31ToPintsel

DMA0 Input trigger.

enumerator kINPUTMUX_FlexSpiRxToDma0
enumerator kINPUTMUX_FlexSpiTxToDma0
enumerator kINPUTMUX_PinInt0ToDma0
enumerator kINPUTMUX_PinInt1ToDma0
enumerator kINPUTMUX_PinInt2ToDma0
enumerator kINPUTMUX_PinInt3ToDma0
enumerator kINPUTMUX_Ctimer0M0ToDma0
enumerator kINPUTMUX_Ctimer0M1ToDma0
enumerator kINPUTMUX_Ctimer1M0ToDma0
enumerator kINPUTMUX_Ctimer1M1ToDma0
enumerator kINPUTMUX_Ctimer2M0ToDma0
enumerator kINPUTMUX_Ctimer2M1ToDma0
enumerator kINPUTMUX_Ctimer3M0ToDma0
enumerator kINPUTMUX_Ctimer3M1ToDma0
enumerator kINPUTMUX_Ctimer4M0ToDma0
enumerator kINPUTMUX_Ctimer4M1ToDma0
enumerator kINPUTMUX_CompOutToDma0
enumerator kINPUTMUX_OtrigAToDma0
enumerator kINPUTMUX_OtrigBToDma0
enumerator kINPUTMUX_OtrigCToDma0

enumerator kINPUTMUX_OtrigDToDma0
enumerator kINPUTMUX_SctDma0ToDma0
enumerator kINPUTMUX_SctDma1ToDma0
enumerator kINPUTMUX_Adc0Tcomp0ToDma0
enumerator kINPUTMUX_Adc1Tcomp0ToDma0
enumerator kINPUTMUX_Hscmp0ToDma0
enumerator kINPUTMUX_Hscmp1ToDma0
enumerator kINPUTMUX_Hscmp2ToDma0
enumerator kINPUTMUX_Aoi0Out0ToDma0
enumerator kINPUTMUX_Aoi0Out1ToDma0
enumerator kINPUTMUX_Aoi0Out2ToDma0
enumerator kINPUTMUX_Aoi0Out3ToDma0
enumerator kINPUTMUX_Aoi1Out0ToDma0
enumerator kINPUTMUX_Aoi1Out1ToDma0
enumerator kINPUTMUX_Aoi1Out2ToDma0
enumerator kINPUTMUX_Aoi1Out3ToDma0
enumerator kINPUTMUX_FlexPwm0ReqCapt0ToDma0
enumerator kINPUTMUX_FlexPwm0ReqCapt1ToDma0
enumerator kINPUTMUX_FlexPwm0ReqCapt2ToDma0
enumerator kINPUTMUX_FlexPwm0ReqCapt3ToDma0
enumerator kINPUTMUX_FlexPwm0ReqVal0ToDma0
enumerator kINPUTMUX_FlexPwm0ReqVal1ToDma0
enumerator kINPUTMUX_FlexPwm0ReqVal2ToDma0
enumerator kINPUTMUX_FlexPwm0ReqVal3ToDma0
enumerator kINPUTMUX_FlexPwm1ReqCapt0ToDma0
enumerator kINPUTMUX_FlexPwm1ReqCapt1ToDma0
enumerator kINPUTMUX_FlexPwm1ReqCapt2ToDma0
enumerator kINPUTMUX_FlexPwm1ReqCapt3ToDma0
enumerator kINPUTMUX_FlexPwm1ReqVal0ToDma0
enumerator kINPUTMUX_FlexPwm1ReqVal1ToDma0
enumerator kINPUTMUX_FlexPwm1ReqVal2ToDma0
enumerator kINPUTMUX_FlexPwm1ReqVal3ToDma0
enumerator kINPUTMUX_TmprOutToDma0

DMA0 output trigger.

enumerator kINPUTMUX__Dma0FlexSpiRxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexSpiTxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma0PinInt0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0PinInt1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0PinInt2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0PinInt3TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer0M0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer0M1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer1M0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer1M1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer2M0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer2M1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer3M0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer3M1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer4M0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Ctimer4M1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0CompOutTrigoutToTriginChannels
enumerator kINPUTMUX__Dma0OtrigATrigoutToTriginChannels
enumerator kINPUTMUX__Dma0OtrigBTrigoutToTriginChannels
enumerator kINPUTMUX__Dma0OtrigCTrigoutToTriginChannels
enumerator kINPUTMUX__Dma0OtrigDTrigoutToTriginChannels
enumerator kINPUTMUX__Dma0SctDma0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0SctDma1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Adc0Tcomp0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Adc1Tcomp0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Hscmp0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Hscmp1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Hscmp2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Aoi0Out0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Aoi0Out1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Aoi0Out2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Aoi0Out3TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Aoi1Out0TrigoutToTriginChannels

enumerator kINPUTMUX__Dma0Aoi1Out1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Aoi1Out2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0Aoi1Out3TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqCapt0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqCapt1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqCapt2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqCapt3TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqVal0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqVal1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqVal2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm0ReqVal3TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqCapt0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqCapt1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqCapt2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqCapt3TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqVal0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqVal1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqVal2TrigoutToTriginChannels
enumerator kINPUTMUX__Dma0FlexPwm1ReqVal3TrigoutToTriginChannels
enumerator kINPUTMUX__TmprOutTrigoutToTriginChannels

Selection for frequency measurement reference clock.

enumerator kINPUTMUX__Xtal32MhzToFreqmeasRef
enumerator kINPUTMUX__FroOsc12MhzToFreqmeasRef
enumerator kINPUTMUX__FroOsc96MhzToFreqmeasRef
enumerator kINPUTMUX__WdoscToFreqmeasRef
enumerator kINPUTMUX__32KhzOscToFreqmeasRef
enumerator kINPUTMUX__MainSysClockToFreqmeasRef
enumerator kINPUTMUX__FreqmeGpioClkAToFreqmeasRef
enumerator kINPUTMUX__FreqmeGpioClkBToFreqmeasRef
enumerator kINPUTMUX__Aoi0Out2ToFreqmeasRef
enumerator kINPUTMUX__Aoi1Out2ToFreqmeasRef

Selection for frequency measurement target clock.

enumerator kINPUTMUX__Xtal32MhzToFreqmeasTarget
enumerator kINPUTMUX__FroOsc12MhzToFreqmeasTarget


```
enumerator kINPUTMUX__FroOsc96MhzToFreqmeasTarget
enumerator kINPUTMUX__WdoscToFreqmeasTarget
enumerator kINPUTMUX__32KhzOscToFreqmeasTarget
enumerator kINPUTMUX__MainSysClockToFreqmeasTarget
enumerator kINPUTMUX__FreqmeGpioClkAToFreqmeasTarget
enumerator kINPUTMUX__FreqmeGpioClkBToFreqmeasTarget
enumerator kINPUTMUX__Aoi0Out2ToFreqmeasTarget
enumerator kINPUTMUX__Aoi1Out2ToFreqmeasTarget
    TIMER3 CAPTSEL.
enumerator kINPUTMUX__CtimerInp0ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp1ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp2ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp3ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp4ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp5ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp6ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp7ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp8ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp9ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp10ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp11ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp12ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp13ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp14ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp15ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp16ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp17ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp18ToTimer3Cptsel
enumerator kINPUTMUX__CtimerInp19ToTimer3Cptsel
enumerator kINPUTMUX__Usb0FrameToggleToTimer3Cptsel
enumerator kINPUTMUX__CompOutToTimer3Cptsel
enumerator kINPUTMUX__I2sSharedWs0ToTimer3Cptsel
enumerator kINPUTMUX__I2sSharedWs1ToTimer3Cptsel
```

enumerator kINPUTMUX__Adc0IrqToTimer3Cptsel
enumerator kINPUTMUX__Adc1IrqToTimer3Cptsel
enumerator kINPUTMUX__Hscmp0OutToTimer3Cptsel
enumerator kINPUTMUX__Hscmp1OutToTimer3Cptsel
enumerator kINPUTMUX__Hscmp2OutToTimer3Cptsel
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToTimer3Cptsel
enumerator kINPUTMUX__Enc0CmpPosMatchToTimer3Cptsel
enumerator kINPUTMUX__Enc1CmpPosMatchToTimer3Cptsel
enumerator kINPUTMUX__Aoi0Out0ToTimer3Cptsel
enumerator kINPUTMUX__Aoi0Out1ToTimer3Cptsel
enumerator kINPUTMUX__Aoi0Out2ToTimer3Cptsel
enumerator kINPUTMUX__Aoi0Out3ToTimer3Cptsel
enumerator kINPUTMUX__Aoi1Out0ToTimer3Cptsel
enumerator kINPUTMUX__Aoi1Out1ToTimer3Cptsel
enumerator kINPUTMUX__Aoi1Out2ToTimer3Cptsel
enumerator kINPUTMUX__Aoi1Out3ToTimer3Cptsel
enumerator kINPUTMUX__TmprOutToTimer3Cptsel
 TIMER3 Trigger.
enumerator kINPUTMUX__CtimerInp0ToTimer3Trigger
enumerator kINPUTMUX__CtimerInp1ToTimer3Trigger
enumerator kINPUTMUX__CtimerInp2ToTimer3Trigger
enumerator kINPUTMUX__CtimerInp3ToTimer3Trigger
enumerator kINPUTMUX__CtimerInp4ToTimer3Trigger
enumerator kINPUTMUX__CtimerInp5ToTimer3Trigger
enumerator kINPUTMUX__CtimerInp6ToTimer3Trigger
enumerator kINPUTMUX__CtimerInp7ToTimer3Trigger

enumerator kINPUTMUX_CtimerInp8ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp9ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp10ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp11ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp12ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp13ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp14ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp15ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp16ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp17ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp18ToTimer3Trigger
enumerator kINPUTMUX_CtimerInp19ToTimer3Trigger
enumerator kINPUTMUX_Usb0FrameToggleToTimer3Trigger
enumerator kINPUTMUX_CompOutToTimer3Trigger
enumerator kINPUTMUX_I2sSharedWs0ToTimer3Trigger
enumerator kINPUTMUX_I2sSharedWs1ToTimer3Trigger
enumerator kINPUTMUX_Adc0IrqToTimer3Trigger
enumerator kINPUTMUX_Adc1IrqToTimer3Trigger
enumerator kINPUTMUX_Hscmp0OutToTimer3Trigger
enumerator kINPUTMUX_Hscmp1OutToTimer3Trigger
enumerator kINPUTMUX_Hscmp2OutToTimer3Trigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig01ToTimer3Trigger
enumerator kINPUTMUX_Enc0CmpPosMatchToTimer3Trigger
enumerator kINPUTMUX_Enc1CmpPosMatchToTimer3Trigger
enumerator kINPUTMUX_Aoi0Out0ToTimer3Trigger
enumerator kINPUTMUX_Aoi0Out1ToTimer3Trigger

enumerator kINPUTMUX__Aoi0Out2ToTimer3Trigger
enumerator kINPUTMUX__Aoi0Out3ToTimer3Trigger
enumerator kINPUTMUX__Aoi1Out0ToTimer3Trigger
enumerator kINPUTMUX__Aoi1Out1ToTimer3Trigger
enumerator kINPUTMUX__Aoi1Out2ToTimer3Trigger
enumerator kINPUTMUX__Aoi1Out3ToTimer3Trigger
enumerator kINPUTMUX__TmprOutToTimer3Trigger
Timer4 CAPTSEL.
enumerator kINPUTMUX__CtimerInp0ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp1ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp2ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp3ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp4ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp5ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp6ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp7ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp8ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp9ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp10ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp11ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp12ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp13ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp14ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp15ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp16ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp17ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp18ToTimer4Captsel
enumerator kINPUTMUX__CtimerInp19ToTimer4Captsel
enumerator kINPUTMUX__Usb0FrameToggleToTimer4Captsel
enumerator kINPUTMUX__CompOutToTimer4Captsel
enumerator kINPUTMUX__I2sSharedWs0ToTimer4Captsel
enumerator kINPUTMUX__I2sSharedWs1ToTimer4Captsel
enumerator kINPUTMUX__Adc0IrqToTimer4Captsel

enumerator kINPUTMUX__Adc1IrqToTimer4Captsel
enumerator kINPUTMUX__Hscmp0OutToTimer4Captsel
enumerator kINPUTMUX__Hscmp1OutToTimer4Captsel
enumerator kINPUTMUX__Hscmp2OutToTimer4Captsel
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToTimer4Captsel
enumerator kINPUTMUX__Enc0CmpPosMatchToTimer4Captsel
enumerator kINPUTMUX__Enc1CmpPosMatchToTimer4Captsel
enumerator kINPUTMUX__Aoi0Out0ToTimer4Captsel
enumerator kINPUTMUX__Aoi0Out1ToTimer4Captsel
enumerator kINPUTMUX__Aoi0Out2ToTimer4Captsel
enumerator kINPUTMUX__Aoi0Out3ToTimer4Captsel
enumerator kINPUTMUX__Aoi1Out0ToTimer4Captsel
enumerator kINPUTMUX__Aoi1Out1ToTimer4Captsel
enumerator kINPUTMUX__Aoi1Out2ToTimer4Captsel
enumerator kINPUTMUX__Aoi1Out3ToTimer4Captsel
enumerator kINPUTMUX__TmprOutToTimer4Captsel
 TIMER4 Trigger.
enumerator kINPUTMUX__CtimerInp0ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp1ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp2ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp3ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp4ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp5ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp6ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp7ToTimer4Trigger
enumerator kINPUTMUX__CtimerInp8ToTimer4Trigger

enumerator kINPUTMUX_CtimerInp9ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp10ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp11ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp12ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp13ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp14ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp15ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp16ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp17ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp18ToTimer4Trigger
enumerator kINPUTMUX_CtimerInp19ToTimer4Trigger
enumerator kINPUTMUX_Usb0FrameToggleToTimer4Trigger
enumerator kINPUTMUX_CompOutToTimer4Trigger
enumerator kINPUTMUX_I2sSharedWs0ToTimer4Trigger
enumerator kINPUTMUX_I2sSharedWs1ToTimer4Trigger
enumerator kINPUTMUX_Adc0IrqToTimer4Trigger
enumerator kINPUTMUX_Adc1IrqToTimer4Trigger
enumerator kINPUTMUX_Hscmp0OutToTimer4Trigger
enumerator kINPUTMUX_Hscmp1OutToTimer4Trigger
enumerator kINPUTMUX_Hscmp2OutToTimer4Trigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig01ToTimer4Trigger
enumerator kINPUTMUX_Enc0CmpPosMatchToTimer4Trigger
enumerator kINPUTMUX_Enc1CmpPosMatchToTimer4Trigger
enumerator kINPUTMUX_Aoi0Out0ToTimer4Trigger
enumerator kINPUTMUX_Aoi0Out1ToTimer4Trigger
enumerator kINPUTMUX_Aoi0Out2ToTimer4Trigger

enumerator kINPUTMUX__Aoi0Out3ToTimer4Trigger
enumerator kINPUTMUX__Aoi1Out0ToTimer4Trigger
enumerator kINPUTMUX__Aoi1Out1ToTimer4Trigger
enumerator kINPUTMUX__Aoi1Out2ToTimer4Trigger
enumerator kINPUTMUX__Aoi1Out3ToTimer4Trigger
enumerator kINPUTMUX__TmprOutToTimer4Trigger
enumerator kINPUTMUX__GpioPort0Pin0ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin1ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin2ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin3ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin4ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin5ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin6ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin7ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin8ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin9ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin10ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin11ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin12ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin13ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin14ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin15ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin16ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin17ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin18ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin19ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin20ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin21ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin22ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin23ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin24ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin25ToPintSecsel
enumerator kINPUTMUX__GpioPort0Pin26ToPintSecsel

enumerator kINPUTMUX_GpioPort0Pin27ToPintSecsel

enumerator kINPUTMUX_GpioPort0Pin28ToPintSecsel

enumerator kINPUTMUX_GpioPort0Pin29ToPintSecsel

enumerator kINPUTMUX_GpioPort0Pin30ToPintSecsel

enumerator kINPUTMUX_GpioPort0Pin31ToPintSecsel

DMA1 Input trigger.

enumerator kINPUTMUX_PinInt0ToDma1

enumerator kINPUTMUX_PinInt1ToDma1

enumerator kINPUTMUX_PinInt2ToDma1

enumerator kINPUTMUX_PinInt3ToDma1

enumerator kINPUTMUX_Ctimer0M0ToDma1

enumerator kINPUTMUX_Ctimer0M1ToDma1

enumerator kINPUTMUX_Ctimer2M0ToDma1

enumerator kINPUTMUX_Ctimer4M0ToDma1

enumerator kINPUTMUX_OtrigAToDma1

enumerator kINPUTMUX_OtrigBToDma1

enumerator kINPUTMUX_OtrigCToDma1

enumerator kINPUTMUX_OtrigDToDma1

enumerator kINPUTMUX_Sct0DmaReq0ToDma1

enumerator kINPUTMUX_Sct0DmaReq1ToDma1

enumerator kINPUTMUX_FlexSpiRxToDma1

enumerator kINPUTMUX_FlexSpiTxToDma1

enumerator kINPUTMUX_Aoi0Out0ToDma1

enumerator kINPUTMUX_Aoi0Out1ToDma1

enumerator kINPUTMUX_Aoi0Out2ToDma1

enumerator kINPUTMUX_Aoi0Out3ToDma1

enumerator kINPUTMUX_Aoi1Out0ToDma1

enumerator kINPUTMUX_Aoi1Out1ToDma1

enumerator kINPUTMUX_Aoi1Out2ToDma1

enumerator kINPUTMUX_Aoi1Out3ToDma1

enumerator kINPUTMUX_TmprOutToDma1

DMA1 output trigger.

enumerator kINPUTMUX_Dma1HsLspiRxTrigoutToTriginChannels

enumerator kINPUTMUX_Dma1HsLspiTxTrigoutToTriginChannels

enumerator kINPUTMUX__Dma1Flexcom0RxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma1Flexcom0TxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma1Flexcom1RxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma1Flexcom1TxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma1Flexcom3RxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma1Flexcom3TxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma1Dmic0Ch0TrigoutToTriginChannels
enumerator kINPUTMUX__Dma1Dmic0Ch1TrigoutToTriginChannels
enumerator kINPUTMUX__Dma1I3c0RxTrigoutToTriginChannels
enumerator kINPUTMUX__Dma1I3c0TxTrigoutToTriginChannels
HSCMP0 trigger.
enumerator kINPUTMUX__PinInt0ToHscmp0Trigger
enumerator kINPUTMUX__PinInt6ToHscmp0Trigger
enumerator kINPUTMUX__SctOut4ToHscmp0Trigger
enumerator kINPUTMUX__SctOut5ToHscmp0Trigger
enumerator kINPUTMUX__SctOut6ToHscmp0Trigger
enumerator kINPUTMUX__Ctimer0M3ToHscmp0Trigger
enumerator kINPUTMUX__Ctimer1M3ToHscmp0Trigger
enumerator kINPUTMUX__Ctimer2M3ToHscmp0Trigger
enumerator kINPUTMUX__Ctimer0M0ToHscmp0Trigger
enumerator kINPUTMUX__Ctimer4M0ToHscmp0Trigger
enumerator kINPUTMUX__ArmTxevToHscmp0Trigger
enumerator kINPUTMUX__GpioIntBmatchToHscmp0Trigger
enumerator kINPUTMUX__Adc0Tcomp0ToHscmp0Trigger
enumerator kINPUTMUX__Adc1Tcomp0ToHscmp0Trigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToHscmp0Trigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToHscmp0Trigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToHscmp0Trigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToHscmp0Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToHscmp0Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToHscmp0Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToHscmp0Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToHscmp0Trigger

enumerator kINPUTMUX__Enc0CmpPosMatchToHscmp0Trigger
enumerator kINPUTMUX__Enc1CmpPosMatchToHscmp0Trigger
enumerator kINPUTMUX__Aoi0Out0ToHscmp0Trigger
enumerator kINPUTMUX__Aoi0Out1ToHscmp0Trigger
enumerator kINPUTMUX__Aoi0Out2ToHscmp0Trigger
enumerator kINPUTMUX__Aoi0Out3ToHscmp0Trigger
enumerator kINPUTMUX__Aoi1Out0ToHscmp0Trigger
enumerator kINPUTMUX__Aoi1Out1ToHscmp0Trigger
enumerator kINPUTMUX__Aoi1Out2ToHscmp0Trigger
enumerator kINPUTMUX__Aoi1Out3ToHscmp0Trigger
enumerator kINPUTMUX__Dma0Trigout0ToHscmp0Trigger
enumerator kINPUTMUX__Dma0Trigout1ToHscmp0Trigger
enumerator kINPUTMUX__Dma0Trigout2ToHscmp0Trigger
ADC0 trigger.
enumerator kINPUTMUX__PinInt0ToAdc0Trigger
enumerator kINPUTMUX__PinInt1ToAdc0Trigger
enumerator kINPUTMUX__SctOut4ToAdc0Trigger
enumerator kINPUTMUX__SctOut5ToAdc0Trigger
enumerator kINPUTMUX__SctOut9ToAdc0Trigger
enumerator kINPUTMUX__Ctimer0M3ToAdc0Trigger
enumerator kINPUTMUX__Ctimer1M3ToAdc0Trigger
enumerator kINPUTMUX__Ctimer2M3ToAdc0Trigger
enumerator kINPUTMUX__Ctimer3M3ToAdc0Trigger
enumerator kINPUTMUX__Ctimer4M3ToAdc0Trigger
enumerator kINPUTMUX__CompOutToAdc0Trigger
enumerator kINPUTMUX__ArmTxevToAdc0Trigger
enumerator kINPUTMUX__GpioIntBmatchToAdc0Trigger
enumerator kINPUTMUX__Adc0Tcomp0ToAdc0Trigger
enumerator kINPUTMUX__Adc0Tcomp1ToAdc0Trigger
enumerator kINPUTMUX__Adc0Tcomp2ToAdc0Trigger
enumerator kINPUTMUX__Adc0Tcomp3ToAdc0Trigger
enumerator kINPUTMUX__Adc1Tcomp0ToAdc0Trigger
enumerator kINPUTMUX__Adc1Tcomp1ToAdc0Trigger

enumerator kINPUTMUX__Adc1Tcomp2ToAdc0Trigger
enumerator kINPUTMUX__Adc1Tcomp3ToAdc0Trigger
enumerator kINPUTMUX__Hscmp0OutToAdc0Trigger
enumerator kINPUTMUX__Hscmp1OutToAdc0Trigger
enumerator kINPUTMUX__Hscmp2OutToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToAdc0Trigger
enumerator kINPUTMUX__Enc0CmpPosMatchToAdc0Trigger
enumerator kINPUTMUX__Enc1CmpPosMatchToAdc0Trigger
enumerator kINPUTMUX__Aoi0Out0ToAdc0Trigger
enumerator kINPUTMUX__Aoi0Out1ToAdc0Trigger
enumerator kINPUTMUX__Aoi0Out2ToAdc0Trigger
enumerator kINPUTMUX__Aoi0Out3ToAdc0Trigger
enumerator kINPUTMUX__Aoi1Out0ToAdc0Trigger
enumerator kINPUTMUX__Aoi1Out1ToAdc0Trigger
enumerator kINPUTMUX__Aoi1Out2ToAdc0Trigger
enumerator kINPUTMUX__Aoi1Out3ToAdc0Trigger
enumerator kINPUTMUX__Dma0Trigout0ToAdc0Trigger
enumerator kINPUTMUX__Dma0Trigout1ToAdc0Trigger

enumerator kINPUTMUX_Dma0Trigout2ToAdc0Trigger
ADC1 trigger.

enumerator kINPUTMUX_PinInt0ToAdc1Trigger

enumerator kINPUTMUX_PinInt2ToAdc1Trigger

enumerator kINPUTMUX_SctOut4ToAdc1Trigger

enumerator kINPUTMUX_SctOut5ToAdc1Trigger

enumerator kINPUTMUX_SctOut3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer0M3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer1M3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer2M3ToAdc1Trigger

enumerator kINPUTMUX_Ctimer3M2ToAdc1Trigger

enumerator kINPUTMUX_Ctimer4M1ToAdc1Trigger

enumerator kINPUTMUX_CompOutToAdc1Trigger

enumerator kINPUTMUX_ArmTxevToAdc1Trigger

enumerator kINPUTMUX_GpioIntBmatchToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp2ToAdc1Trigger

enumerator kINPUTMUX_Adc0Tcomp3ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp0ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp1ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp2ToAdc1Trigger

enumerator kINPUTMUX_Adc1Tcomp3ToAdc1Trigger

enumerator kINPUTMUX_Hscmp0OutToAdc1Trigger

enumerator kINPUTMUX_Hscmp1OutToAdc1Trigger

enumerator kINPUTMUX_Hscmp2OutToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm0MuxTrig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm0MuxTrig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm1MuxTrig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm1MuxTrig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm2MuxTrig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm2MuxTrig1ToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm3MuxTrig0ToAdc1Trigger

enumerator kINPUTMUX_Pwm0Sm3MuxTrig1ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig0ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig1ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig0ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig1ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig0ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig1ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig0ToAdc1Trigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig1ToAdc1Trigger
enumerator kINPUTMUX_Enc0CmpPosMatchToAdc1Trigger
enumerator kINPUTMUX_Enc1CmpPosMatchToAdc1Trigger
enumerator kINPUTMUX_Aoi0Out0ToAdc1Trigger
enumerator kINPUTMUX_Aoi0Out1ToAdc1Trigger
enumerator kINPUTMUX_Aoi0Out2ToAdc1Trigger
enumerator kINPUTMUX_Aoi0Out3ToAdc1Trigger
enumerator kINPUTMUX_Aoi1Out0ToAdc1Trigger
enumerator kINPUTMUX_Aoi1Out1ToAdc1Trigger
enumerator kINPUTMUX_Aoi1Out2ToAdc1Trigger
enumerator kINPUTMUX_Aoi1Out3ToAdc1Trigger
enumerator kINPUTMUX_Dma0Trigout0ToAdc1Trigger
enumerator kINPUTMUX_Dma0Trigout1ToAdc1Trigger
enumerator kINPUTMUX_Dma0Trigout2ToAdc1Trigger
DAC0 trigger.
enumerator kINPUTMUX_PinInt0ToDac0Trigger
enumerator kINPUTMUX_PinInt3ToDac0Trigger
enumerator kINPUTMUX_SctOut4ToDac0Trigger
enumerator kINPUTMUX_SctOut5ToDac0Trigger
enumerator kINPUTMUX_SctOut0ToDac0Trigger
enumerator kINPUTMUX_Ctimer0M3ToDac0Trigger
enumerator kINPUTMUX_Ctimer1M3ToDac0Trigger
enumerator kINPUTMUX_Ctimer2M3ToDac0Trigger
enumerator kINPUTMUX_Ctimer2M0ToDac0Trigger
enumerator kINPUTMUX_Ctimer3M0ToDac0Trigger

enumerator kINPUTMUX__CompOutToDac0Trigger
enumerator kINPUTMUX__ArmTxevToDac0Trigger
enumerator kINPUTMUX__GppointBmatchToDac0Trigger
enumerator kINPUTMUX__Adc0Tcomp0ToDac0Trigger
enumerator kINPUTMUX__Adc1Tcomp0ToDac0Trigger
enumerator kINPUTMUX__Hscmp0OutToDac0Trigger
enumerator kINPUTMUX__Hscmp1OutToDac0Trigger
enumerator kINPUTMUX__Hscmp2OutToDac0Trigger
enumerator kINPUTMUX__Aoi0Out0ToDac0Trigger
enumerator kINPUTMUX__Aoi0Out1ToDac0Trigger
enumerator kINPUTMUX__Aoi0Out2ToDac0Trigger
enumerator kINPUTMUX__Aoi0Out3ToDac0Trigger
enumerator kINPUTMUX__Aoi1Out0ToDac0Trigger
enumerator kINPUTMUX__Aoi1Out1ToDac0Trigger
enumerator kINPUTMUX__Aoi1Out2ToDac0Trigger
enumerator kINPUTMUX__Aoi1Out3ToDac0Trigger
enumerator kINPUTMUX__Dma0Trigout0ToDac0Trigger
enumerator kINPUTMUX__Dma0Trigout1ToDac0Trigger
enumerator kINPUTMUX__Dma0Trigout2ToDac0Trigger
DAC1 trigger.
enumerator kINPUTMUX__PinInt0ToDac1Trigger
enumerator kINPUTMUX__PinInt4ToDac1Trigger
enumerator kINPUTMUX__SctOut4ToDac1Trigger
enumerator kINPUTMUX__SctOut5ToDac1Trigger
enumerator kINPUTMUX__SctOut1ToDac1Trigger
enumerator kINPUTMUX__Ctimer0M3ToDac1Trigger
enumerator kINPUTMUX__Ctimer1M3ToDac1Trigger
enumerator kINPUTMUX__Ctimer2M3ToDac1Trigger
enumerator kINPUTMUX__Ctimer2M1ToDac1Trigger
enumerator kINPUTMUX__Ctimer3M1ToDac1Trigger
enumerator kINPUTMUX__CompOutToDac1Trigger
enumerator kINPUTMUX__ArmTxevToDac1Trigger
enumerator kINPUTMUX__GppointBmatchToDac1Trigger

enumerator kINPUTMUX__Adc0Tcomp1ToDac1Trigger
enumerator kINPUTMUX__Adc1Tcomp1ToDac1Trigger
enumerator kINPUTMUX__Hscmp0OutToDac1Trigger
enumerator kINPUTMUX__Hscmp1OutToDac1Trigger
enumerator kINPUTMUX__Hscmp2OutToDac1Trigger
enumerator kINPUTMUX__Aoi0Out0ToDac1Trigger
enumerator kINPUTMUX__Aoi0Out1ToDac1Trigger
enumerator kINPUTMUX__Aoi0Out2ToDac1Trigger
enumerator kINPUTMUX__Aoi0Out3ToDac1Trigger
enumerator kINPUTMUX__Aoi1Out0ToDac1Trigger
enumerator kINPUTMUX__Aoi1Out1ToDac1Trigger
enumerator kINPUTMUX__Aoi1Out2ToDac1Trigger
enumerator kINPUTMUX__Aoi1Out3ToDac1Trigger
enumerator kINPUTMUX__Dma0Trigout0ToDac1Trigger
enumerator kINPUTMUX__Dma0Trigout1ToDac1Trigger
enumerator kINPUTMUX__Dma0Trigout2ToDac1Trigger
DAC2 trigger.

enumerator kINPUTMUX__PinInt0ToDac2Trigger
enumerator kINPUTMUX__PinInt5ToDac2Trigger
enumerator kINPUTMUX__SctOut4ToDac2Trigger
enumerator kINPUTMUX__SctOut5ToDac2Trigger
enumerator kINPUTMUX__SctOut2ToDac2Trigger
enumerator kINPUTMUX__Ctimer0M3ToDac2Trigger
enumerator kINPUTMUX__Ctimer1M3ToDac2Trigger
enumerator kINPUTMUX__Ctimer2M3ToDac2Trigger
enumerator kINPUTMUX__Ctimer2M2ToDac2Trigger
enumerator kINPUTMUX__Ctimer3M2ToDac2Trigger
enumerator kINPUTMUX__CompOutToDac2Trigger
enumerator kINPUTMUX__ArmTxevToDac2Trigger
enumerator kINPUTMUX__GpioIntBmatchToDac2Trigger
enumerator kINPUTMUX__Adc0Tcomp2ToDac2Trigger
enumerator kINPUTMUX__Adc1Tcomp2ToDac2Trigger
enumerator kINPUTMUX__Hscmp0OutToDac2Trigger

enumerator kINPUTMUX__Hscmp1OutToDac2Trigger
enumerator kINPUTMUX__Hscmp2OutToDac2Trigger
enumerator kINPUTMUX__Aoi0Out0ToDac2Trigger
enumerator kINPUTMUX__Aoi0Out1ToDac2Trigger
enumerator kINPUTMUX__Aoi0Out2ToDac2Trigger
enumerator kINPUTMUX__Aoi0Out3ToDac2Trigger
enumerator kINPUTMUX__Aoi1Out0ToDac2Trigger
enumerator kINPUTMUX__Aoi1Out1ToDac2Trigger
enumerator kINPUTMUX__Aoi1Out2ToDac2Trigger
enumerator kINPUTMUX__Aoi1Out3ToDac2Trigger
enumerator kINPUTMUX__Dma0Trigout0ToDac2Trigger
enumerator kINPUTMUX__Dma0Trigout1ToDac2Trigger
enumerator kINPUTMUX__Dma0Trigout2ToDac2Trigger
ENC0 TRIG.

enumerator kINPUTMUX__PinInt0ToEnc0Trigger
enumerator kINPUTMUX__PinInt4ToEnc0Trigger
enumerator kINPUTMUX__SctOut4ToEnc0Trigger
enumerator kINPUTMUX__SctOut5ToEnc0Trigger
enumerator kINPUTMUX__SctOut1ToEnc0Trigger
enumerator kINPUTMUX__Ctimer0M3ToEnc0Trigger
enumerator kINPUTMUX__Ctimer1M3ToEnc0Trigger
enumerator kINPUTMUX__Ctimer2M3ToEnc0Trigger
enumerator kINPUTMUX__Ctimer1M0ToEnc0Trigger
enumerator kINPUTMUX__Ctimer3M0ToEnc0Trigger
enumerator kINPUTMUX__CompOutToEnc0Trigger
enumerator kINPUTMUX__ArmTxevToEnc0Trigger
enumerator kINPUTMUX__GppointBmatchToEnc0Trigger
enumerator kINPUTMUX__Adc0Tcomp0ToEnc0Trigger
enumerator kINPUTMUX__Adc0Tcomp1ToEnc0Trigger
enumerator kINPUTMUX__Adc0Tcomp2ToEnc0Trigger
enumerator kINPUTMUX__Adc0Tcomp3ToEnc0Trigger
enumerator kINPUTMUX__Adc1Tcomp0ToEnc0Trigger
enumerator kINPUTMUX__Adc1Tcomp1ToEnc0Trigger

enumerator kINPUTMUX__Adc1Tcomp2ToEnc0Trigger
enumerator kINPUTMUX__Adc1Tcomp3ToEnc0Trigger
enumerator kINPUTMUX__Hscmp0OutToEnc0Trigger
enumerator kINPUTMUX__Hscmp1OutToEnc0Trigger
enumerator kINPUTMUX__Hscmp2OutToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc0Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc0Trigger
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc0Trigger
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc0Trigger
enumerator kINPUTMUX__Aoi0Out0ToEnc0Trigger
enumerator kINPUTMUX__Aoi0Out1ToEnc0Trigger
enumerator kINPUTMUX__Aoi0Out2ToEnc0Trigger
enumerator kINPUTMUX__Aoi0Out3ToEnc0Trigger
enumerator kINPUTMUX__Aoi1Out0ToEnc0Trigger
enumerator kINPUTMUX__Aoi1Out1ToEnc0Trigger
enumerator kINPUTMUX__Aoi1Out2ToEnc0Trigger
enumerator kINPUTMUX__Aoi1Out3ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn0ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn1ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn2ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn3ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn4ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn5ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn6ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn7ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn8ToEnc0Trigger
enumerator kINPUTMUX__ExtTrigIn9ToEnc0Trigger

enumerator kINPUTMUX__Dma0Trigout0ToEnc0Trigger
enumerator kINPUTMUX__Dma0Trigout1ToEnc0Trigger
enumerator kINPUTMUX__Dma0Trigout2ToEnc0Trigger
ENC0 HOME.
enumerator kINPUTMUX__PinInt0ToEnc0Home
enumerator kINPUTMUX__PinInt4ToEnc0Home
enumerator kINPUTMUX__SctOut4ToEnc0Home
enumerator kINPUTMUX__SctOut5ToEnc0Home
enumerator kINPUTMUX__SctOut1ToEnc0Home
enumerator kINPUTMUX__Ctimer0M3ToEnc0Home
enumerator kINPUTMUX__Ctimer1M3ToEnc0Home
enumerator kINPUTMUX__Ctimer2M3ToEnc0Home
enumerator kINPUTMUX__Ctimer1M0ToEnc0Home
enumerator kINPUTMUX__Ctimer3M0ToEnc0Home
enumerator kINPUTMUX__CompOutToEnc0Home
enumerator kINPUTMUX__ArmTxevToEnc0Home
enumerator kINPUTMUX__GpioIntBmatchToEnc0Home
enumerator kINPUTMUX__Adc0Tcomp0ToEnc0Home
enumerator kINPUTMUX__Adc0Tcomp1ToEnc0Home
enumerator kINPUTMUX__Adc0Tcomp2ToEnc0Home
enumerator kINPUTMUX__Adc0Tcomp3ToEnc0Home
enumerator kINPUTMUX__Adc1Tcomp0ToEnc0Home
enumerator kINPUTMUX__Adc1Tcomp1ToEnc0Home
enumerator kINPUTMUX__Adc1Tcomp2ToEnc0Home
enumerator kINPUTMUX__Adc1Tcomp3ToEnc0Home
enumerator kINPUTMUX__Hscmp0OutToEnc0Home
enumerator kINPUTMUX__Hscmp1OutToEnc0Home
enumerator kINPUTMUX__Hscmp2OutToEnc0Home
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc0Home
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc0Home
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc0Home
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc0Home
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc0Home

enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc0Home
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc0Home
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc0Home
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc0Home
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc0Home
enumerator kINPUTMUX__Aoi0Out0ToEnc0Home
enumerator kINPUTMUX__Aoi0Out1ToEnc0Home
enumerator kINPUTMUX__Aoi0Out2ToEnc0Home
enumerator kINPUTMUX__Aoi0Out3ToEnc0Home
enumerator kINPUTMUX__Aoi1Out0ToEnc0Home
enumerator kINPUTMUX__Aoi1Out1ToEnc0Home
enumerator kINPUTMUX__Aoi1Out2ToEnc0Home
enumerator kINPUTMUX__Aoi1Out3ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn0ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn1ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn2ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn3ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn4ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn5ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn6ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn7ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn8ToEnc0Home
enumerator kINPUTMUX__ExtTrigIn9ToEnc0Home
enumerator kINPUTMUX__Dma0Trigout0ToEnc0Home
enumerator kINPUTMUX__Dma0Trigout1ToEnc0Home
enumerator kINPUTMUX__Dma0Trigout2ToEnc0Home
ENC0 INDEX.
enumerator kINPUTMUX__PinInt0ToEnc0Index
enumerator kINPUTMUX__PinInt4ToEnc0Index
enumerator kINPUTMUX__SctOut4ToEnc0Index
enumerator kINPUTMUX__SctOut5ToEnc0Index
enumerator kINPUTMUX__SctOut1ToEnc0Index
enumerator kINPUTMUX__Ctimer0M3ToEnc0Index

enumerator kINPUTMUX__Ctimer1M3ToEnc0Index
enumerator kINPUTMUX__Ctimer2M3ToEnc0Index
enumerator kINPUTMUX__Ctimer1M0ToEnc0Index
enumerator kINPUTMUX__Ctimer3M0ToEnc0Index
enumerator kINPUTMUX__CompOutToEnc0Index
enumerator kINPUTMUX__ArmTxevToEnc0Index
enumerator kINPUTMUX__GpjointBmatchToEnc0Index
enumerator kINPUTMUX__Adc0Tcomp0ToEnc0Index
enumerator kINPUTMUX__Adc0Tcomp1ToEnc0Index
enumerator kINPUTMUX__Adc0Tcomp2ToEnc0Index
enumerator kINPUTMUX__Adc0Tcomp3ToEnc0Index
enumerator kINPUTMUX__Adc1Tcomp0ToEnc0Index
enumerator kINPUTMUX__Adc1Tcomp1ToEnc0Index
enumerator kINPUTMUX__Adc1Tcomp2ToEnc0Index
enumerator kINPUTMUX__Adc1Tcomp3ToEnc0Index
enumerator kINPUTMUX__Hscmp0OutToEnc0Index
enumerator kINPUTMUX__Hscmp1OutToEnc0Index
enumerator kINPUTMUX__Hscmp2OutToEnc0Index
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc0Index
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc0Index
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc0Index
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc0Index
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc0Index
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc0Index
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc0Index
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc0Index
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc0Index
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc0Index
enumerator kINPUTMUX__Aoi0Out0ToEnc0Index
enumerator kINPUTMUX__Aoi0Out1ToEnc0Index
enumerator kINPUTMUX__Aoi0Out2ToEnc0Index
enumerator kINPUTMUX__Aoi0Out3ToEnc0Index
enumerator kINPUTMUX__Aoi1Out0ToEnc0Index

enumerator kINPUTMUX__Aoi1Out1ToEnc0Index
enumerator kINPUTMUX__Aoi1Out2ToEnc0Index
enumerator kINPUTMUX__Aoi1Out3ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn0ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn1ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn2ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn3ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn4ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn5ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn6ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn7ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn8ToEnc0Index
enumerator kINPUTMUX__ExtTrigIn9ToEnc0Index
enumerator kINPUTMUX__Dma0Trigout0ToEnc0Index
enumerator kINPUTMUX__Dma0Trigout1ToEnc0Index
enumerator kINPUTMUX__Dma0Trigout2ToEnc0Index

ENC0 PHASEB.

enumerator kINPUTMUX__PinInt0ToEnc0Phaseb
enumerator kINPUTMUX__PinInt4ToEnc0Phaseb
enumerator kINPUTMUX__SctOut4ToEnc0Phaseb
enumerator kINPUTMUX__SctOut5ToEnc0Phaseb
enumerator kINPUTMUX__SctOut1ToEnc0Phaseb
enumerator kINPUTMUX__Ctimer0M3ToEnc0Phaseb
enumerator kINPUTMUX__Ctimer1M3ToEnc0Phaseb
enumerator kINPUTMUX__Ctimer2M3ToEnc0Phaseb
enumerator kINPUTMUX__Ctimer1M0ToEnc0Phaseb
enumerator kINPUTMUX__Ctimer3M0ToEnc0Phaseb
enumerator kINPUTMUX__CompOutToEnc0Phaseb
enumerator kINPUTMUX__ArmTxevToEnc0Phaseb
enumerator kINPUTMUX__GpioIntBmatchToEnc0Phaseb
enumerator kINPUTMUX__Adc0Tcomp0ToEnc0Phaseb
enumerator kINPUTMUX__Adc0Tcomp1ToEnc0Phaseb
enumerator kINPUTMUX__Adc0Tcomp2ToEnc0Phaseb

enumerator kINPUTMUX__Adc0Tcomp3ToEnc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp0ToEnc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp1ToEnc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp2ToEnc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp3ToEnc0Phaseb
enumerator kINPUTMUX__Hscmp0OutToEnc0Phaseb
enumerator kINPUTMUX__Hscmp1OutToEnc0Phaseb
enumerator kINPUTMUX__Hscmp2OutToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc0Phaseb
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc0Phaseb
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc0Phaseb
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc0Phaseb
enumerator kINPUTMUX__Aoi0Out0ToEnc0Phaseb
enumerator kINPUTMUX__Aoi0Out1ToEnc0Phaseb
enumerator kINPUTMUX__Aoi0Out2ToEnc0Phaseb
enumerator kINPUTMUX__Aoi0Out3ToEnc0Phaseb
enumerator kINPUTMUX__Aoi1Out0ToEnc0Phaseb
enumerator kINPUTMUX__Aoi1Out1ToEnc0Phaseb
enumerator kINPUTMUX__Aoi1Out2ToEnc0Phaseb
enumerator kINPUTMUX__Aoi1Out3ToEnc0Phaseb
enumerator kINPUTMUX__ExtTrigIn0ToEnc0Phaseb
enumerator kINPUTMUX__ExtTrigIn1ToEnc0Phaseb
enumerator kINPUTMUX__ExtTrigIn2ToEnc0Phaseb
enumerator kINPUTMUX__ExtTrigIn3ToEnc0Phaseb
enumerator kINPUTMUX__ExtTrigIn4ToEnc0Phaseb
enumerator kINPUTMUX__ExtTrigIn5ToEnc0Phaseb
enumerator kINPUTMUX__ExtTrigIn6ToEnc0Phaseb

enumerator kINPUTMUX_ExtTrigIn7ToEnc0Phaseb
enumerator kINPUTMUX_ExtTrigIn8ToEnc0Phaseb
enumerator kINPUTMUX_ExtTrigIn9ToEnc0Phaseb
enumerator kINPUTMUX_Dma0Trigout0ToEnc0Phaseb
enumerator kINPUTMUX_Dma0Trigout1ToEnc0Phaseb
enumerator kINPUTMUX_Dma0Trigout2ToEnc0Phaseb
ENC0 PHASEA.
enumerator kINPUTMUX_PinInt0ToEnc0Phasea
enumerator kINPUTMUX_PinInt4ToEnc0Phasea
enumerator kINPUTMUX_SctOut4ToEnc0Phasea
enumerator kINPUTMUX_SctOut5ToEnc0Phasea
enumerator kINPUTMUX_SctOut1ToEnc0Phasea
enumerator kINPUTMUX_Ctimer0M3ToEnc0Phasea
enumerator kINPUTMUX_Ctimer1M3ToEnc0Phasea
enumerator kINPUTMUX_Ctimer2M3ToEnc0Phasea
enumerator kINPUTMUX_Ctimer1M0ToEnc0Phasea
enumerator kINPUTMUX_Ctimer3M0ToEnc0Phasea
enumerator kINPUTMUX_CompOutToEnc0Phasea
enumerator kINPUTMUX_ArmTxevToEnc0Phasea
enumerator kINPUTMUX_GpjointBmatchToEnc0Phasea
enumerator kINPUTMUX_Adc0Tcomp0ToEnc0Phasea
enumerator kINPUTMUX_Adc0Tcomp1ToEnc0Phasea
enumerator kINPUTMUX_Adc0Tcomp2ToEnc0Phasea
enumerator kINPUTMUX_Adc0Tcomp3ToEnc0Phasea
enumerator kINPUTMUX_Adc1Tcomp0ToEnc0Phasea
enumerator kINPUTMUX_Adc1Tcomp1ToEnc0Phasea
enumerator kINPUTMUX_Adc1Tcomp2ToEnc0Phasea
enumerator kINPUTMUX_Adc1Tcomp3ToEnc0Phasea
enumerator kINPUTMUX_Hscmp0OutToEnc0Phasea
enumerator kINPUTMUX_Hscmp1OutToEnc0Phasea
enumerator kINPUTMUX_Hscmp2OutToEnc0Phasea
enumerator kINPUTMUX_Pwm1Sm0MuxTrig0ToEnc0Phasea
enumerator kINPUTMUX_Pwm1Sm0MuxTrig1ToEnc0Phasea

enumerator kINPUTMUX_Pwm1Sm1MuxTrig0ToEnc0Phasea
enumerator kINPUTMUX_Pwm1Sm1MuxTrig1ToEnc0Phasea
enumerator kINPUTMUX_Pwm1Sm2MuxTrig0ToEnc0Phasea
enumerator kINPUTMUX_Pwm1Sm2MuxTrig1ToEnc0Phasea
enumerator kINPUTMUX_Pwm1Sm3MuxTrig0ToEnc0Phasea
enumerator kINPUTMUX_Pwm1Sm3MuxTrig1ToEnc0Phasea
enumerator kINPUTMUX_Enc0CmpPosMatchToEnc0Phasea
enumerator kINPUTMUX_Enc1CmpPosMatchToEnc0Phasea
enumerator kINPUTMUX_Aoi0Out0ToEnc0Phasea
enumerator kINPUTMUX_Aoi0Out1ToEnc0Phasea
enumerator kINPUTMUX_Aoi0Out2ToEnc0Phasea
enumerator kINPUTMUX_Aoi0Out3ToEnc0Phasea
enumerator kINPUTMUX_Aoi1Out0ToEnc0Phasea
enumerator kINPUTMUX_Aoi1Out1ToEnc0Phasea
enumerator kINPUTMUX_Aoi1Out2ToEnc0Phasea
enumerator kINPUTMUX_Aoi1Out3ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn0ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn1ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn2ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn3ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn4ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn5ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn6ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn7ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn8ToEnc0Phasea
enumerator kINPUTMUX_ExtTrigIn9ToEnc0Phasea
enumerator kINPUTMUX_Dma0Trigout0ToEnc0Phasea
enumerator kINPUTMUX_Dma0Trigout1ToEnc0Phasea
enumerator kINPUTMUX_Dma0Trigout2ToEnc0Phasea
ENC1 TRIG.
enumerator kINPUTMUX_PinInt0ToEnc1Trigger
enumerator kINPUTMUX_PinInt5ToEnc1Trigger
enumerator kINPUTMUX_SctOut4ToEnc1Trigger

enumerator kINPUTMUX__SctOut5ToEnc1Trigger
enumerator kINPUTMUX__SctOut7ToEnc1Trigger
enumerator kINPUTMUX__Ctimer0M3ToEnc1Trigger
enumerator kINPUTMUX__Ctimer1M3ToEnc1Trigger
enumerator kINPUTMUX__Ctimer2M3ToEnc1Trigger
enumerator kINPUTMUX__Ctimer1M1ToEnc1Trigger
enumerator kINPUTMUX__Ctimer3M1ToEnc1Trigger
enumerator kINPUTMUX__CompOutToEnc1Trigger
enumerator kINPUTMUX__ArmTxevToEnc1Trigger
enumerator kINPUTMUX__GpioIntBmatchToEnc1Trigger
enumerator kINPUTMUX__Adc0Tcomp0ToEnc1Trigger
enumerator kINPUTMUX__Adc0Tcomp1ToEnc1Trigger
enumerator kINPUTMUX__Adc0Tcomp2ToEnc1Trigger
enumerator kINPUTMUX__Adc0Tcomp3ToEnc1Trigger
enumerator kINPUTMUX__Adc1Tcomp0ToEnc1Trigger
enumerator kINPUTMUX__Adc1Tcomp1ToEnc1Trigger
enumerator kINPUTMUX__Adc1Tcomp2ToEnc1Trigger
enumerator kINPUTMUX__Adc1Tcomp3ToEnc1Trigger
enumerator kINPUTMUX__Hscmp0OutToEnc1Trigger
enumerator kINPUTMUX__Hscmp1OutToEnc1Trigger
enumerator kINPUTMUX__Hscmp2OutToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc1Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc1Trigger
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc1Trigger
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc1Trigger
enumerator kINPUTMUX__Aoi0Out0ToEnc1Trigger
enumerator kINPUTMUX__Aoi0Out1ToEnc1Trigger

enumerator kINPUTMUX__Aoi0Out2ToEnc1Trigger
enumerator kINPUTMUX__Aoi0Out3ToEnc1Trigger
enumerator kINPUTMUX__Aoi1Out0ToEnc1Trigger
enumerator kINPUTMUX__Aoi1Out1ToEnc1Trigger
enumerator kINPUTMUX__Aoi1Out2ToEnc1Trigger
enumerator kINPUTMUX__Aoi1Out3ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn0ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn1ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn2ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn3ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn4ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn5ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn6ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn7ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn8ToEnc1Trigger
enumerator kINPUTMUX__ExtTrigIn9ToEnc1Trigger
enumerator kINPUTMUX__Dma0Trigout0ToEnc1Trigger
enumerator kINPUTMUX__Dma0Trigout1ToEnc1Trigger
enumerator kINPUTMUX__Dma0Trigout2ToEnc1Trigger
ENC1 HOME.
enumerator kINPUTMUX__PinInt0ToEnc1Home
enumerator kINPUTMUX__PinInt5ToEnc1Home
enumerator kINPUTMUX__SctOut4ToEnc1Home
enumerator kINPUTMUX__SctOut5ToEnc1Home
enumerator kINPUTMUX__SctOut7ToEnc1Home
enumerator kINPUTMUX__Ctimer0M3ToEnc1Home
enumerator kINPUTMUX__Ctimer1M3ToEnc1Home
enumerator kINPUTMUX__Ctimer2M3ToEnc1Home
enumerator kINPUTMUX__Ctimer1M1ToEnc1Home
enumerator kINPUTMUX__Ctimer3M1ToEnc1Home
enumerator kINPUTMUX__CompOutToEnc1Home
enumerator kINPUTMUX__ArmTxevToEnc1Home
enumerator kINPUTMUX__GpjointBmatchToEnc1Home

enumerator kINPUTMUX__Adc0Tcomp0ToEnc1Home
enumerator kINPUTMUX__Adc0Tcomp1ToEnc1Home
enumerator kINPUTMUX__Adc0Tcomp2ToEnc1Home
enumerator kINPUTMUX__Adc0Tcomp3ToEnc1Home
enumerator kINPUTMUX__Adc1Tcomp0ToEnc1Home
enumerator kINPUTMUX__Adc1Tcomp1ToEnc1Home
enumerator kINPUTMUX__Adc1Tcomp2ToEnc1Home
enumerator kINPUTMUX__Adc1Tcomp3ToEnc1Home
enumerator kINPUTMUX__Hscmp0OutToEnc1Home
enumerator kINPUTMUX__Hscmp1OutToEnc1Home
enumerator kINPUTMUX__Hscmp2OutToEnc1Home
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc1Home
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc1Home
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc1Home
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc1Home
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc1Home
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc1Home
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc1Home
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc1Home
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc1Home
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc1Home
enumerator kINPUTMUX__Aoi0Out0ToEnc1Home
enumerator kINPUTMUX__Aoi0Out1ToEnc1Home
enumerator kINPUTMUX__Aoi0Out2ToEnc1Home
enumerator kINPUTMUX__Aoi0Out3ToEnc1Home
enumerator kINPUTMUX__Aoi1Out0ToEnc1Home
enumerator kINPUTMUX__Aoi1Out1ToEnc1Home
enumerator kINPUTMUX__Aoi1Out2ToEnc1Home
enumerator kINPUTMUX__Aoi1Out3ToEnc1Home
enumerator kINPUTMUX__ExtTrigIn0ToEnc1Home
enumerator kINPUTMUX__ExtTrigIn1ToEnc1Home
enumerator kINPUTMUX__ExtTrigIn2ToEnc1Home
enumerator kINPUTMUX__ExtTrigIn3ToEnc1Home

enumerator kINPUTMUX_ExtTrigIn4ToEnc1Home
enumerator kINPUTMUX_ExtTrigIn5ToEnc1Home
enumerator kINPUTMUX_ExtTrigIn6ToEnc1Home
enumerator kINPUTMUX_ExtTrigIn7ToEnc1Home
enumerator kINPUTMUX_ExtTrigIn8ToEnc1Home
enumerator kINPUTMUX_ExtTrigIn9ToEnc1Home
enumerator kINPUTMUX_Dma0Trigout0ToEnc1Home
enumerator kINPUTMUX_Dma0Trigout1ToEnc1Home
enumerator kINPUTMUX_Dma0Trigout2ToEnc1Home
ENC1 INDEX.

enumerator kINPUTMUX_PinInt0ToEnc1Index
enumerator kINPUTMUX_PinInt5ToEnc1Index
enumerator kINPUTMUX_SctOut4ToEnc1Index
enumerator kINPUTMUX_SctOut5ToEnc1Index
enumerator kINPUTMUX_SctOut7ToEnc1Index
enumerator kINPUTMUX_Ctimer0M3ToEnc1Index
enumerator kINPUTMUX_Ctimer1M3ToEnc1Index
enumerator kINPUTMUX_Ctimer2M3ToEnc1Index
enumerator kINPUTMUX_Ctimer1M1ToEnc1Index
enumerator kINPUTMUX_Ctimer3M1ToEnc1Index
enumerator kINPUTMUX_CompOutToEnc1Index
enumerator kINPUTMUX_ArmTxevToEnc1Index
enumerator kINPUTMUX_GppointBmatchToEnc1Index
enumerator kINPUTMUX_Adc0Tcomp0ToEnc1Index
enumerator kINPUTMUX_Adc0Tcomp1ToEnc1Index
enumerator kINPUTMUX_Adc0Tcomp2ToEnc1Index
enumerator kINPUTMUX_Adc0Tcomp3ToEnc1Index
enumerator kINPUTMUX_Adc1Tcomp0ToEnc1Index
enumerator kINPUTMUX_Adc1Tcomp1ToEnc1Index
enumerator kINPUTMUX_Adc1Tcomp2ToEnc1Index
enumerator kINPUTMUX_Adc1Tcomp3ToEnc1Index
enumerator kINPUTMUX_Hscmp0OutToEnc1Index
enumerator kINPUTMUX_Hscmp1OutToEnc1Index

enumerator kINPUTMUX__Hscmp2OutToEnc1Index
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc1Index
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc1Index
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc1Index
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc1Index
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc1Index
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc1Index
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc1Index
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc1Index
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc1Index
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc1Index
enumerator kINPUTMUX__Aoi0Out0ToEnc1Index
enumerator kINPUTMUX__Aoi0Out1ToEnc1Index
enumerator kINPUTMUX__Aoi0Out2ToEnc1Index
enumerator kINPUTMUX__Aoi0Out3ToEnc1Index
enumerator kINPUTMUX__Aoi1Out0ToEnc1Index
enumerator kINPUTMUX__Aoi1Out1ToEnc1Index
enumerator kINPUTMUX__Aoi1Out2ToEnc1Index
enumerator kINPUTMUX__Aoi1Out3ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn0ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn1ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn2ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn3ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn4ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn5ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn6ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn7ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn8ToEnc1Index
enumerator kINPUTMUX__ExtTrigIn9ToEnc1Index
enumerator kINPUTMUX__Dma0Trigout0ToEnc1Index
enumerator kINPUTMUX__Dma0Trigout1ToEnc1Index
enumerator kINPUTMUX__Dma0Trigout2ToEnc1Index
ENC1 PHASEB.
enumerator kINPUTMUX__PinInt0ToEnc1Phaseb

enumerator kINPUTMUX_PinInt5ToEnc1Phaseb
enumerator kINPUTMUX_SctOut4ToEnc1Phaseb
enumerator kINPUTMUX_SctOut5ToEnc1Phaseb
enumerator kINPUTMUX_SctOut7ToEnc1Phaseb
enumerator kINPUTMUX_Ctimer0M3ToEnc1Phaseb
enumerator kINPUTMUX_Ctimer1M3ToEnc1Phaseb
enumerator kINPUTMUX_Ctimer2M3ToEnc1Phaseb
enumerator kINPUTMUX_Ctimer1M1ToEnc1Phaseb
enumerator kINPUTMUX_Ctimer3M1ToEnc1Phaseb
enumerator kINPUTMUX_CompOutToEnc1Phaseb
enumerator kINPUTMUX_ArmTxevToEnc1Phaseb
enumerator kINPUTMUX_GpjointBmatchToEnc1Phaseb
enumerator kINPUTMUX_Adc0Tcomp0ToEnc1Phaseb
enumerator kINPUTMUX_Adc0Tcomp1ToEnc1Phaseb
enumerator kINPUTMUX_Adc0Tcomp2ToEnc1Phaseb
enumerator kINPUTMUX_Adc0Tcomp3ToEnc1Phaseb
enumerator kINPUTMUX_Adc1Tcomp0ToEnc1Phaseb
enumerator kINPUTMUX_Adc1Tcomp1ToEnc1Phaseb
enumerator kINPUTMUX_Adc1Tcomp2ToEnc1Phaseb
enumerator kINPUTMUX_Adc1Tcomp3ToEnc1Phaseb
enumerator kINPUTMUX_Hscmp0OutToEnc1Phaseb
enumerator kINPUTMUX_Hscmp1OutToEnc1Phaseb
enumerator kINPUTMUX_Hscmp2OutToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm0MuxTrig0ToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm0MuxTrig1ToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm1MuxTrig0ToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm1MuxTrig1ToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm2MuxTrig0ToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm2MuxTrig1ToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm3MuxTrig0ToEnc1Phaseb
enumerator kINPUTMUX_Pwm1Sm3MuxTrig1ToEnc1Phaseb
enumerator kINPUTMUX_Enc0CmpPosMatchToEnc1Phaseb
enumerator kINPUTMUX_Enc1CmpPosMatchToEnc1Phaseb

enumerator kINPUTMUX__Aoi0Out0ToEnc1Phaseb
enumerator kINPUTMUX__Aoi0Out1ToEnc1Phaseb
enumerator kINPUTMUX__Aoi0Out2ToEnc1Phaseb
enumerator kINPUTMUX__Aoi0Out3ToEnc1Phaseb
enumerator kINPUTMUX__Aoi1Out0ToEnc1Phaseb
enumerator kINPUTMUX__Aoi1Out1ToEnc1Phaseb
enumerator kINPUTMUX__Aoi1Out2ToEnc1Phaseb
enumerator kINPUTMUX__Aoi1Out3ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn0ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn1ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn2ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn3ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn4ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn5ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn6ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn7ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn8ToEnc1Phaseb
enumerator kINPUTMUX__ExtTrigIn9ToEnc1Phaseb
enumerator kINPUTMUX__Dma0Trigout0ToEnc1Phaseb
enumerator kINPUTMUX__Dma0Trigout1ToEnc1Phaseb
enumerator kINPUTMUX__Dma0Trigout2ToEnc1Phaseb
ENC1 PHASEA.
enumerator kINPUTMUX__PinInt0ToEnc1Phasea
enumerator kINPUTMUX__PinInt5ToEnc1Phasea
enumerator kINPUTMUX__SctOut4ToEnc1Phasea
enumerator kINPUTMUX__SctOut5ToEnc1Phasea
enumerator kINPUTMUX__SctOut7ToEnc1Phasea
enumerator kINPUTMUX__Ctimer0M3ToEnc1Phasea
enumerator kINPUTMUX__Ctimer1M3ToEnc1Phasea
enumerator kINPUTMUX__Ctimer2M3ToEnc1Phasea
enumerator kINPUTMUX__Ctimer1M1ToEnc1Phasea
enumerator kINPUTMUX__Ctimer3M1ToEnc1Phasea
enumerator kINPUTMUX__CompOutToEnc1Phasea

enumerator kINPUTMUX__ArmTxeVToEnc1Phasea
enumerator kINPUTMUX__GpioIntBmatchToEnc1Phasea
enumerator kINPUTMUX__Adc0Tcomp0ToEnc1Phasea
enumerator kINPUTMUX__Adc0Tcomp1ToEnc1Phasea
enumerator kINPUTMUX__Adc0Tcomp2ToEnc1Phasea
enumerator kINPUTMUX__Adc0Tcomp3ToEnc1Phasea
enumerator kINPUTMUX__Adc1Tcomp0ToEnc1Phasea
enumerator kINPUTMUX__Adc1Tcomp1ToEnc1Phasea
enumerator kINPUTMUX__Adc1Tcomp2ToEnc1Phasea
enumerator kINPUTMUX__Adc1Tcomp3ToEnc1Phasea
enumerator kINPUTMUX__Hscmp0OutToEnc1Phasea
enumerator kINPUTMUX__Hscmp1OutToEnc1Phasea
enumerator kINPUTMUX__Hscmp2OutToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToEnc1Phasea
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToEnc1Phasea
enumerator kINPUTMUX__Enc0CmpPosMatchToEnc1Phasea
enumerator kINPUTMUX__Enc1CmpPosMatchToEnc1Phasea
enumerator kINPUTMUX__Aoi0Out0ToEnc1Phasea
enumerator kINPUTMUX__Aoi0Out1ToEnc1Phasea
enumerator kINPUTMUX__Aoi0Out2ToEnc1Phasea
enumerator kINPUTMUX__Aoi0Out3ToEnc1Phasea
enumerator kINPUTMUX__Aoi1Out0ToEnc1Phasea
enumerator kINPUTMUX__Aoi1Out1ToEnc1Phasea
enumerator kINPUTMUX__Aoi1Out2ToEnc1Phasea
enumerator kINPUTMUX__Aoi1Out3ToEnc1Phasea
enumerator kINPUTMUX__ExtTrigIn0ToEnc1Phasea
enumerator kINPUTMUX__ExtTrigIn1ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn2ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn3ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn4ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn5ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn6ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn7ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn8ToEnc1Phasea

enumerator kINPUTMUX_ExtTrigIn9ToEnc1Phasea

enumerator kINPUTMUX_Dma0Trigout0ToEnc1Phasea

enumerator kINPUTMUX_Dma0Trigout1ToEnc1Phasea

enumerator kINPUTMUX_Dma0Trigout2ToEnc1Phasea

PWM0 external synchronization trigger.

enumerator kINPUTMUX_PinInt0ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_PinInt5ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_SctOut4ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_SctOut5ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_SctOut2ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Ctimer0M3ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Ctimer1M3ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Ctimer2M3ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Ctimer2M0ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Ctimer4M0ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_CompOutToPwm0ExtSyncTrigger

enumerator kINPUTMUX_ArmTxevToPwm0ExtSyncTrigger

enumerator kINPUTMUX_GppointBmatchToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc0Tcomp0ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc0Tcomp1ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc0Tcomp2ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc0Tcomp3ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc1Tcomp0ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc1Tcomp1ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc1Tcomp2ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Adc1Tcomp3ToPwm0ExtSyncTrigger

enumerator kINPUTMUX__Hscmp0OutToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Hscmp1OutToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Hscmp2OutToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Enc0CmpPosMatchToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Enc1CmpPosMatchToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi0Out0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi0Out1ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi0Out2ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi0Out3ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi1Out0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi1Out1ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi1Out2ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Aoi1Out3ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn1ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn2ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn3ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn4ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn5ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn6ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn7ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn8ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn9ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Dma0Trigout0ToPwm0ExtSyncTrigger
enumerator kINPUTMUX__Dma0Trigout1ToPwm0ExtSyncTrigger

enumerator kINPUTMUX_Dma0Trigout2ToPwm0ExtSyncTrigger

PWM0 input trigger connections trigger.

enumerator kINPUTMUX_PinInt0ToPwm0ExtATrigger

enumerator kINPUTMUX_PinInt5ToPwm0ExtATrigger

enumerator kINPUTMUX_SctOut4ToPwm0ExtATrigger

enumerator kINPUTMUX_SctOut5ToPwm0ExtATrigger

enumerator kINPUTMUX_SctOut2ToPwm0ExtATrigger

enumerator kINPUTMUX_Ctimer0M3ToPwm0ExtATrigger

enumerator kINPUTMUX_Ctimer1M3ToPwm0ExtATrigger

enumerator kINPUTMUX_Ctimer2M3ToPwm0ExtATrigger

enumerator kINPUTMUX_Ctimer2M0ToPwm0ExtATrigger

enumerator kINPUTMUX_Ctimer4M0ToPwm0ExtATrigger

enumerator kINPUTMUX_CompOutToPwm0ExtATrigger

enumerator kINPUTMUX_ArmTxevToPwm0ExtATrigger

enumerator kINPUTMUX_GpioIntBmatchToPwm0ExtATrigger

enumerator kINPUTMUX_Adc0Tcomp0ToPwm0ExtATrigger

enumerator kINPUTMUX_Adc0Tcomp1ToPwm0ExtATrigger

enumerator kINPUTMUX_Adc0Tcomp2ToPwm0ExtATrigger

enumerator kINPUTMUX_Adc0Tcomp3ToPwm0ExtATrigger

enumerator kINPUTMUX_Adc1Tcomp0ToPwm0ExtATrigger

enumerator kINPUTMUX_Adc1Tcomp1ToPwm0ExtATrigger

enumerator kINPUTMUX_Adc1Tcomp2ToPwm0ExtATrigger

enumerator kINPUTMUX_Adc1Tcomp3ToPwm0ExtATrigger

enumerator kINPUTMUX_Hscmp0OutToPwm0ExtATrigger

enumerator kINPUTMUX_Hscmp1OutToPwm0ExtATrigger

enumerator kINPUTMUX_Hscmp2OutToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm0MuxTrig0ToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm0MuxTrig1ToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm1MuxTrig0ToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm1MuxTrig1ToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm2MuxTrig0ToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm2MuxTrig1ToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm3MuxTrig0ToPwm0ExtATrigger

enumerator kINPUTMUX_Pwm1Sm3MuxTrig1ToPwm0ExtATrigger
enumerator kINPUTMUX_Enc0CmpPosMatchToPwm0ExtATrigger
enumerator kINPUTMUX_Enc1CmpPosMatchToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi0Out0ToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi0Out1ToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi0Out2ToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi0Out3ToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi1Out0ToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi1Out1ToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi1Out2ToPwm0ExtATrigger
enumerator kINPUTMUX_Aoi1Out3ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn0ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn1ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn2ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn3ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn4ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn5ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn6ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn7ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn8ToPwm0ExtATrigger
enumerator kINPUTMUX_ExtTrigIn9ToPwm0ExtATrigger
enumerator kINPUTMUX_Dma0Trigout0ToPwm0ExtATrigger
enumerator kINPUTMUX_Dma0Trigout1ToPwm0ExtATrigger
enumerator kINPUTMUX_Dma0Trigout2ToPwm0ExtATrigger
PWM0 external force trigger connections trigger.
enumerator kINPUTMUX_PinInt0ToPwm0ExtForceTrigger
enumerator kINPUTMUX_PinInt5ToPwm0ExtForceTrigger
enumerator kINPUTMUX_SctOut4ToPwm0ExtForceTrigger
enumerator kINPUTMUX_SctOut5ToPwm0ExtForceTrigger
enumerator kINPUTMUX_SctOut2ToPwm0ExtForceTrigger
enumerator kINPUTMUX_Ctimer0M3ToPwm0ExtForceTrigger
enumerator kINPUTMUX_Ctimer1M3ToPwm0ExtForceTrigger
enumerator kINPUTMUX_Ctimer2M3ToPwm0ExtForceTrigger

enumerator kINPUTMUX__Ctimer2M0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Ctimer4M0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__CompOutToPwm0ExtForceTrigger
enumerator kINPUTMUX__ArmTxevToPwm0ExtForceTrigger
enumerator kINPUTMUX__GpjointBmatchToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc0Tcomp0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc0Tcomp1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc0Tcomp2ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc0Tcomp3ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc1Tcomp0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc1Tcomp1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc1Tcomp2ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Adc1Tcomp3ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Hscmp0OutToPwm0ExtForceTrigger
enumerator kINPUTMUX__Hscmp1OutToPwm0ExtForceTrigger
enumerator kINPUTMUX__Hscmp2OutToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Enc0CmpPosMatchToPwm0ExtForceTrigger
enumerator kINPUTMUX__Enc1CmpPosMatchToPwm0ExtForceTrigger
enumerator kINPUTMUX__Aoi0Out0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Aoi0Out1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Aoi0Out2ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Aoi0Out3ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Aoi1Out0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Aoi1Out1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Aoi1Out2ToPwm0ExtForceTrigger

enumerator kINPUTMUX__Aoi1Out3ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn2ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn3ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn4ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn5ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn6ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn7ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn8ToPwm0ExtForceTrigger
enumerator kINPUTMUX__ExtTrigIn9ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Dma0Trigout0ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Dma0Trigout1ToPwm0ExtForceTrigger
enumerator kINPUTMUX__Dma0Trigout2ToPwm0ExtForceTrigger
PWM0 fault input trigger connections trigger.
enumerator kINPUTMUX__PinInt0ToPwm0FaultTrigger
enumerator kINPUTMUX__PinInt5ToPwm0FaultTrigger
enumerator kINPUTMUX__SctOut4ToPwm0FaultTrigger
enumerator kINPUTMUX__SctOut5ToPwm0FaultTrigger
enumerator kINPUTMUX__SctOut2ToPwm0FaultTrigger
enumerator kINPUTMUX__Ctimer0M3ToPwm0FaultTrigger
enumerator kINPUTMUX__Ctimer1M3ToPwm0FaultTrigger
enumerator kINPUTMUX__Ctimer2M3ToPwm0FaultTrigger
enumerator kINPUTMUX__Ctimer2M0ToPwm0FaultTrigger
enumerator kINPUTMUX__Ctimer4M0ToPwm0FaultTrigger
enumerator kINPUTMUX__CompOutToPwm0FaultTrigger
enumerator kINPUTMUX__ArmTxevToPwm0FaultTrigger
enumerator kINPUTMUX__GpionBmatchToPwm0FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp0ToPwm0FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp1ToPwm0FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp2ToPwm0FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp3ToPwm0FaultTrigger
enumerator kINPUTMUX__Adc1Tcomp0ToPwm0FaultTrigger

enumerator kINPUTMUX__Adc1Tcomp1ToPwm0FaultTrigger
enumerator kINPUTMUX__Adc1Tcomp2ToPwm0FaultTrigger
enumerator kINPUTMUX__Adc1Tcomp3ToPwm0FaultTrigger
enumerator kINPUTMUX__Hscmp0OutToPwm0FaultTrigger
enumerator kINPUTMUX__Hscmp1OutToPwm0FaultTrigger
enumerator kINPUTMUX__Hscmp2OutToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToPwm0FaultTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToPwm0FaultTrigger
enumerator kINPUTMUX__Enc0CmpPosMatchToPwm0FaultTrigger
enumerator kINPUTMUX__Enc1CmpPosMatchToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi0Out0ToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi0Out1ToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi0Out2ToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi0Out3ToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi1Out0ToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi1Out1ToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi1Out2ToPwm0FaultTrigger
enumerator kINPUTMUX__Aoi1Out3ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn0ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn1ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn2ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn3ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn4ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn5ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn6ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn7ToPwm0FaultTrigger
enumerator kINPUTMUX__ExtTrigIn8ToPwm0FaultTrigger

enumerator kINPUTMUX_ExtTrigIn9ToPwm0FaultTrigger

enumerator kINPUTMUX_Dma0Trigout0ToPwm0FaultTrigger

enumerator kINPUTMUX_Dma0Trigout1ToPwm0FaultTrigger

enumerator kINPUTMUX_Dma0Trigout2ToPwm0FaultTrigger

PWM0 extclk input trigger connections trigger.

enumerator kINPUTMUX_PinInt0ToPwm0ExtClkTrigger

enumerator kINPUTMUX_PinInt5ToPwm0ExtClkTrigger

enumerator kINPUTMUX_SctOut4ToPwm0ExtClkTrigger

enumerator kINPUTMUX_SctOut5ToPwm0ExtClkTrigger

enumerator kINPUTMUX_SctOut2ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Ctimer0M3ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Ctimer1M3ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Ctimer2M3ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Ctimer2M0ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Ctimer4M0ToPwm0ExtClkTrigger

enumerator kINPUTMUX_CompOutToPwm0ExtClkTrigger

enumerator kINPUTMUX_ArmTxevToPwm0ExtClkTrigger

enumerator kINPUTMUX_GppointBmatchToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc0Tcomp0ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc0Tcomp1ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc0Tcomp2ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc0Tcomp3ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc1Tcomp0ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc1Tcomp1ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc1Tcomp2ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Adc1Tcomp3ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Hscmp0OutToPwm0ExtClkTrigger

enumerator kINPUTMUX_Hscmp1OutToPwm0ExtClkTrigger

enumerator kINPUTMUX_Hscmp2OutToPwm0ExtClkTrigger

enumerator kINPUTMUX_Pwm1Sm0MuxTrig0ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Pwm1Sm0MuxTrig1ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Pwm1Sm1MuxTrig0ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Pwm1Sm1MuxTrig1ToPwm0ExtClkTrigger

enumerator kINPUTMUX_Pwm1Sm2MuxTrig0ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig1ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig0ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig1ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Enc0CmpPosMatchToPwm0ExtClkTrigger
enumerator kINPUTMUX_Enc1CmpPosMatchToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out0ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out1ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out2ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out3ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out0ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out1ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out2ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out3ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn0ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn1ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn2ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn3ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn4ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn5ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn6ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn7ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn8ToPwm0ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn9ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Dma0Trigout0ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Dma0Trigout1ToPwm0ExtClkTrigger
enumerator kINPUTMUX_Dma0Trigout2ToPwm0ExtClkTrigger
PWM1 external synchronization trigger.
enumerator kINPUTMUX_PinInt0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_PinInt2ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_SctOut4ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_SctOut5ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_SctOut3ToPwm1ExtSyncTrigger

enumerator kINPUTMUX_Ctimer0M3ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Ctimer1M3ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Ctimer2M3ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Ctimer2M1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Ctimer4M1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_CompOutToPwm1ExtSyncTrigger
enumerator kINPUTMUX_ArmTxevToPwm1ExtSyncTrigger
enumerator kINPUTMUX_GpiontBmatchToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc0Tcomp0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc0Tcomp1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc0Tcomp2ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc0Tcomp3ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc1Tcomp0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc1Tcomp1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc1Tcomp2ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Adc1Tcomp3ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Hscmp0OutToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Hscmp1OutToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Hscmp2OutToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Enc0CmpPosMatchToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Enc1CmpPosMatchToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Aoi0Out0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Aoi0Out1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Aoi0Out2ToPwm1ExtSyncTrigger
enumerator kINPUTMUX_Aoi0Out3ToPwm1ExtSyncTrigger

enumerator kINPUTMUX__Aoi1Out0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__Aoi1Out1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__Aoi1Out2ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__Aoi1Out3ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn2ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn3ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn4ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn5ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn6ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn7ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn8ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__ExtTrigIn9ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__Dma0Trigout0ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__Dma0Trigout1ToPwm1ExtSyncTrigger
enumerator kINPUTMUX__Dma0Trigout2ToPwm1ExtSyncTrigger
PWM1 input trigger connections trigger.
enumerator kINPUTMUX__PinInt0ToPwm1ExtATrigger
enumerator kINPUTMUX__PinInt2ToPwm1ExtATrigger
enumerator kINPUTMUX__SctOut4ToPwm1ExtATrigger
enumerator kINPUTMUX__SctOut5ToPwm1ExtATrigger
enumerator kINPUTMUX__SctOut3ToPwm1ExtATrigger
enumerator kINPUTMUX__Ctimer0M3ToPwm1ExtATrigger
enumerator kINPUTMUX__Ctimer1M3ToPwm1ExtATrigger
enumerator kINPUTMUX__Ctimer2M3ToPwm1ExtATrigger
enumerator kINPUTMUX__Ctimer2M1ToPwm1ExtATrigger
enumerator kINPUTMUX__Ctimer4M1ToPwm1ExtATrigger
enumerator kINPUTMUX__CompOutToPwm1ExtATrigger
enumerator kINPUTMUX__ArmTxevToPwm1ExtATrigger
enumerator kINPUTMUX__GppointBmatchToPwm1ExtATrigger
enumerator kINPUTMUX__Adc0Tcomp0ToPwm1ExtATrigger
enumerator kINPUTMUX__Adc0Tcomp1ToPwm1ExtATrigger

enumerator kINPUTMUX__Adc0Tcomp2ToPwm1ExtATrigger
enumerator kINPUTMUX__Adc0Tcomp3ToPwm1ExtATrigger
enumerator kINPUTMUX__Adc1Tcomp0ToPwm1ExtATrigger
enumerator kINPUTMUX__Adc1Tcomp1ToPwm1ExtATrigger
enumerator kINPUTMUX__Adc1Tcomp2ToPwm1ExtATrigger
enumerator kINPUTMUX__Adc1Tcomp3ToPwm1ExtATrigger
enumerator kINPUTMUX__Hscmp0OutToPwm1ExtATrigger
enumerator kINPUTMUX__Hscmp1OutToPwm1ExtATrigger
enumerator kINPUTMUX__Hscmp2OutToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig0ToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig1ToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig0ToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig1ToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig0ToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig1ToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig0ToPwm1ExtATrigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig1ToPwm1ExtATrigger
enumerator kINPUTMUX__Enc0CmpPosMatchToPwm1ExtATrigger
enumerator kINPUTMUX__Enc1CmpPosMatchToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi0Out0ToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi0Out1ToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi0Out2ToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi0Out3ToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi1Out0ToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi1Out1ToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi1Out2ToPwm1ExtATrigger
enumerator kINPUTMUX__Aoi1Out3ToPwm1ExtATrigger
enumerator kINPUTMUX__ExtTrigIn0ToPwm1ExtATrigger
enumerator kINPUTMUX__ExtTrigIn1ToPwm1ExtATrigger
enumerator kINPUTMUX__ExtTrigIn2ToPwm1ExtATrigger
enumerator kINPUTMUX__ExtTrigIn3ToPwm1ExtATrigger
enumerator kINPUTMUX__ExtTrigIn4ToPwm1ExtATrigger
enumerator kINPUTMUX__ExtTrigIn5ToPwm1ExtATrigger

enumerator kINPUTMUX_ExtTrigIn6ToPwm1ExtATrigger

enumerator kINPUTMUX_ExtTrigIn7ToPwm1ExtATrigger

enumerator kINPUTMUX_ExtTrigIn8ToPwm1ExtATrigger

enumerator kINPUTMUX_ExtTrigIn9ToPwm1ExtATrigger

enumerator kINPUTMUX_Dma0Trigout0ToPwm1ExtATrigger

enumerator kINPUTMUX_Dma0Trigout1ToPwm1ExtATrigger

enumerator kINPUTMUX_Dma0Trigout2ToPwm1ExtATrigger

PWM1 external force trigger connections.

enumerator kINPUTMUX_PinInt0ToPwm1ExtForceTrigger

enumerator kINPUTMUX_PinInt2ToPwm1ExtForceTrigger

enumerator kINPUTMUX_SctOut4ToPwm1ExtForceTrigger

enumerator kINPUTMUX_SctOut5ToPwm1ExtForceTrigger

enumerator kINPUTMUX_SctOut3ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Ctimer0M3ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Ctimer1M3ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Ctimer2M3ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Ctimer2M1ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Ctimer4M1ToPwm1ExtForceTrigger

enumerator kINPUTMUX_CompOutToPwm1ExtForceTrigger

enumerator kINPUTMUX_ArmTxevToPwm1ExtForceTrigger

enumerator kINPUTMUX_GppointBmatchToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc0Tcomp0ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc0Tcomp1ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc0Tcomp2ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc0Tcomp3ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc1Tcomp0ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc1Tcomp1ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc1Tcomp2ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Adc1Tcomp3ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Hscmp0OutToPwm1ExtForceTrigger

enumerator kINPUTMUX_Hscmp1OutToPwm1ExtForceTrigger

enumerator kINPUTMUX_Hscmp2OutToPwm1ExtForceTrigger

enumerator kINPUTMUX_Pwm0Sm0MuxTrig0ToPwm1ExtForceTrigger

enumerator kINPUTMUX_Pwm0Sm0MuxTrig1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig0ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig0ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig0ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Enc0CmpPosMatchToPwm1ExtForceTrigger
enumerator kINPUTMUX_Enc1CmpPosMatchToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi0Out0ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi0Out1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi0Out2ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi0Out3ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi1Out0ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi1Out1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi1Out2ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Aoi1Out3ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn0ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn2ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn3ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn4ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn5ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn6ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn7ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn8ToPwm1ExtForceTrigger
enumerator kINPUTMUX_ExtTrigIn9ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Dma0Trigout0ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Dma0Trigout1ToPwm1ExtForceTrigger
enumerator kINPUTMUX_Dma0Trigout2ToPwm1ExtForceTrigger
PWM1 fault input trigger connections trigger.
enumerator kINPUTMUX_PinInt0ToPwm1FaultTrigger
enumerator kINPUTMUX_PinInt2ToPwm1FaultTrigger

enumerator kINPUTMUX__SctOut4ToPwm1FaultTrigger
enumerator kINPUTMUX__SctOut5ToPwm1FaultTrigger
enumerator kINPUTMUX__SctOut3ToPwm1FaultTrigger
enumerator kINPUTMUX__Ctimer0M3ToPwm1FaultTrigger
enumerator kINPUTMUX__Ctimer1M3ToPwm1FaultTrigger
enumerator kINPUTMUX__Ctimer2M3ToPwm1FaultTrigger
enumerator kINPUTMUX__Ctimer2M1ToPwm1FaultTrigger
enumerator kINPUTMUX__Ctimer4M1ToPwm1FaultTrigger
enumerator kINPUTMUX__CompOutToPwm1FaultTrigger
enumerator kINPUTMUX__ArmTxevToPwm1FaultTrigger
enumerator kINPUTMUX__GpjointBmatchToPwm1FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp0ToPwm1FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp1ToPwm1FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp2ToPwm1FaultTrigger
enumerator kINPUTMUX__Adc0Tcomp3ToPwm1FaultTrigger
enumerator kINPUTMUX__Adc1Tcomp0ToPwm1FaultTrigger
enumerator kINPUTMUX__Adc1Tcomp1ToPwm1FaultTrigger
enumerator kINPUTMUX__Adc1Tcomp2ToPwm1FaultTrigger
enumerator kINPUTMUX__Adc1Tcomp3ToPwm1FaultTrigger
enumerator kINPUTMUX__Hscmp0OutToPwm1FaultTrigger
enumerator kINPUTMUX__Hscmp1OutToPwm1FaultTrigger
enumerator kINPUTMUX__Hscmp2OutToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig0ToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig1ToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig0ToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig1ToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig0ToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig1ToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig0ToPwm1FaultTrigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig1ToPwm1FaultTrigger
enumerator kINPUTMUX__Enc0CmpPosMatchToPwm1FaultTrigger
enumerator kINPUTMUX__Enc1CmpPosMatchToPwm1FaultTrigger
enumerator kINPUTMUX__Aoi0Out0ToPwm1FaultTrigger

enumerator kINPUTMUX__Aoi0Out1ToPwm1FaultTrigger
enumerator kINPUTMUX__Aoi0Out2ToPwm1FaultTrigger
enumerator kINPUTMUX__Aoi0Out3ToPwm1FaultTrigger
enumerator kINPUTMUX__Aoi1Out0ToPwm1FaultTrigger
enumerator kINPUTMUX__Aoi1Out1ToPwm1FaultTrigger
enumerator kINPUTMUX__Aoi1Out2ToPwm1FaultTrigger
enumerator kINPUTMUX__Aoi1Out3ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn0ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn1ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn2ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn3ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn4ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn5ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn6ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn7ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn8ToPwm1FaultTrigger
enumerator kINPUTMUX__ExtTrigIn9ToPwm1FaultTrigger
enumerator kINPUTMUX__Dma0Trigout0ToPwm1FaultTrigger
enumerator kINPUTMUX__Dma0Trigout1ToPwm1FaultTrigger
enumerator kINPUTMUX__Dma0Trigout2ToPwm1FaultTrigger
PWM1 extclk input trigger connections trigger.
enumerator kINPUTMUX__PinInt0ToPwm1ExtClkTrigger
enumerator kINPUTMUX__PinInt2ToPwm1ExtClkTrigger
enumerator kINPUTMUX__SctOut4ToPwm1ExtClkTrigger
enumerator kINPUTMUX__SctOut5ToPwm1ExtClkTrigger
enumerator kINPUTMUX__SctOut3ToPwm1ExtClkTrigger
enumerator kINPUTMUX__Ctimer0M3ToPwm1ExtClkTrigger
enumerator kINPUTMUX__Ctimer1M3ToPwm1ExtClkTrigger
enumerator kINPUTMUX__Ctimer2M3ToPwm1ExtClkTrigger
enumerator kINPUTMUX__Ctimer2M1ToPwm1ExtClkTrigger
enumerator kINPUTMUX__Ctimer4M1ToPwm1ExtClkTrigger
enumerator kINPUTMUX__CompOutToPwm1ExtClkTrigger
enumerator kINPUTMUX__ArmTxevToPwm1ExtClkTrigger

enumerator kINPUTMUX_GpointBmatchToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc0Tcomp0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc0Tcomp1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc0Tcomp2ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc0Tcomp3ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc1Tcomp0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc1Tcomp1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc1Tcomp2ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Adc1Tcomp3ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Hscmp0OutToPwm1ExtClkTrigger
enumerator kINPUTMUX_Hscmp1OutToPwm1ExtClkTrigger
enumerator kINPUTMUX_Hscmp2OutToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Enc0CmpPosMatchToPwm1ExtClkTrigger
enumerator kINPUTMUX_Enc1CmpPosMatchToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out2ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi0Out3ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out2ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Aoi1Out3ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn2ToPwm1ExtClkTrigger

enumerator kINPUTMUX_ExtTrigIn3ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn4ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn5ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn6ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn7ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn8ToPwm1ExtClkTrigger
enumerator kINPUTMUX_ExtTrigIn9ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Dma0Trigout0ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Dma0Trigout1ToPwm1ExtClkTrigger
enumerator kINPUTMUX_Dma0Trigout2ToPwm1ExtClkTrigger
 AOIO trigger.
enumerator kINPUTMUX_PinInt0ToAoi0InTrigger
enumerator kINPUTMUX_PinInt1ToAoi0InTrigger
enumerator kINPUTMUX_SctOut0ToAoi0InTrigger
enumerator kINPUTMUX_SctOut1ToAoi0InTrigger
enumerator kINPUTMUX_SctOut2ToAoi0InTrigger
enumerator kINPUTMUX_SctOut3ToAoi0InTrigger
enumerator kINPUTMUX_Ctimer0M3ToAoi0InTrigger
enumerator kINPUTMUX_Ctimer1M3ToAoi0InTrigger
enumerator kINPUTMUX_Ctimer2M3ToAoi0InTrigger
enumerator kINPUTMUX_Ctimer2M2ToAoi0InTrigger
enumerator kINPUTMUX_Ctimer3M2ToAoi0InTrigger
enumerator kINPUTMUX_Ctimer4M2ToAoi0InTrigger
enumerator kINPUTMUX_CompOutToAoi0InTrigger
enumerator kINPUTMUX_GppointBmatchToAoi0InTrigger
enumerator kINPUTMUX_Adc0IrqToAoi0InTrigger
enumerator kINPUTMUX_Adc1IrqToAoi0InTrigger
enumerator kINPUTMUX_Adc0Tcomp0ToAoi0InTrigger
enumerator kINPUTMUX_Adc0Tcomp1ToAoi0InTrigger
enumerator kINPUTMUX_Adc0Tcomp2ToAoi0InTrigger
enumerator kINPUTMUX_Adc0Tcomp3ToAoi0InTrigger
enumerator kINPUTMUX_Adc1Tcomp0ToAoi0InTrigger
enumerator kINPUTMUX_Adc1Tcomp1ToAoi0InTrigger

enumerator kINPUTMUX__Adc1Tcomp2ToAoi0InTrigger
enumerator kINPUTMUX__Adc1Tcomp3ToAoi0InTrigger
enumerator kINPUTMUX__Hscmp0OutToAoi0InTrigger
enumerator kINPUTMUX__Hscmp1OutToAoi0InTrigger
enumerator kINPUTMUX__Hscmp2OutToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig0ToAoi0InTrigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig1ToAoi0InTrigger
enumerator kINPUTMUX__Enc0CmpPosMatchToAoi0InTrigger
enumerator kINPUTMUX__Enc1CmpPosMatchToAoi0InTrigger
enumerator kINPUTMUX__ExtTrigIn0ToAoi0InTrigger
enumerator kINPUTMUX__ExtTrigIn1ToAoi0InTrigger
enumerator kINPUTMUX__ExtTrigIn2ToAoi0InTrigger
enumerator kINPUTMUX__ExtTrigIn3ToAoi0InTrigger
enumerator kINPUTMUX__Dma0Trigout0ToAoi0InTrigger
enumerator kINPUTMUX__Dma0Trigout1ToAoi0InTrigger
enumerator kINPUTMUX__Dma0Trigout2ToAoi0InTrigger
enumerator kINPUTMUX__Dma0Trigout3ToAoi0InTrigger
enumerator kINPUTMUX__Dma0Trigout4ToAoi0InTrigger
enumerator kINPUTMUX__Dma0Trigout5ToAoi0InTrigger

enumerator kINPUTMUX_Dma0Trigout6ToAoi0InTrigger
enumerator kINPUTMUX_Dma1Trigout0ToAoi0InTrigger
enumerator kINPUTMUX_Dma1Trigout1ToAoi0InTrigger
enumerator kINPUTMUX_Dma1Trigout2ToAoi0InTrigger
AOI1 trigger.
enumerator kINPUTMUX_PinInt0ToAoi1InTrigger
enumerator kINPUTMUX_PinInt1ToAoi1InTrigger
enumerator kINPUTMUX_SctOut0ToAoi1InTrigger
enumerator kINPUTMUX_SctOut1ToAoi1InTrigger
enumerator kINPUTMUX_SctOut2ToAoi1InTrigger
enumerator kINPUTMUX_SctOut3ToAoi1InTrigger
enumerator kINPUTMUX_Ctimer0M3ToAoi1InTrigger
enumerator kINPUTMUX_Ctimer1M3ToAoi1InTrigger
enumerator kINPUTMUX_Ctimer2M3ToAoi1InTrigger
enumerator kINPUTMUX_Ctimer2M2ToAoi1InTrigger
enumerator kINPUTMUX_Ctimer3M2ToAoi1InTrigger
enumerator kINPUTMUX_Ctimer4M2ToAoi1InTrigger
enumerator kINPUTMUX_CompOutToAoi1InTrigger
enumerator kINPUTMUX_GpioIntBmatchToAoi1InTrigger
enumerator kINPUTMUX_Adc0IrqToAoi1InTrigger
enumerator kINPUTMUX_Adc1IrqToAoi1InTrigger
enumerator kINPUTMUX_Adc0Tcomp0ToAoi1InTrigger
enumerator kINPUTMUX_Adc0Tcomp1ToAoi1InTrigger
enumerator kINPUTMUX_Adc0Tcomp2ToAoi1InTrigger
enumerator kINPUTMUX_Adc0Tcomp3ToAoi1InTrigger
enumerator kINPUTMUX_Adc1Tcomp0ToAoi1InTrigger
enumerator kINPUTMUX_Adc1Tcomp1ToAoi1InTrigger
enumerator kINPUTMUX_Adc1Tcomp2ToAoi1InTrigger
enumerator kINPUTMUX_Adc1Tcomp3ToAoi1InTrigger
enumerator kINPUTMUX_Hscmp0OutToAoi1InTrigger
enumerator kINPUTMUX_Hscmp1OutToAoi1InTrigger
enumerator kINPUTMUX_Hscmp2OutToAoi1InTrigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig0ToAoi1InTrigger

enumerator kINPUTMUX_Pwm0Sm0MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig0ToAoi1InTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig0ToAoi1InTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig0ToAoi1InTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig0ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig0ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig0ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig0ToAoi1InTrigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig1ToAoi1InTrigger
enumerator kINPUTMUX_Enc0CmpPosMatchToAoi1InTrigger
enumerator kINPUTMUX_Enc1CmpPosMatchToAoi1InTrigger
enumerator kINPUTMUX_ExtTrigIn0ToAoi1InTrigger
enumerator kINPUTMUX_ExtTrigIn1ToAoi1InTrigger
enumerator kINPUTMUX_ExtTrigIn2ToAoi1InTrigger
enumerator kINPUTMUX_ExtTrigIn3ToAoi1InTrigger
enumerator kINPUTMUX_Dma0Trigout0ToAoi1InTrigger
enumerator kINPUTMUX_Dma0Trigout1ToAoi1InTrigger
enumerator kINPUTMUX_Dma0Trigout2ToAoi1InTrigger
enumerator kINPUTMUX_Dma0Trigout3ToAoi1InTrigger
enumerator kINPUTMUX_Dma0Trigout4ToAoi1InTrigger
enumerator kINPUTMUX_Dma0Trigout5ToAoi1InTrigger
enumerator kINPUTMUX_Dma0Trigout6ToAoi1InTrigger
enumerator kINPUTMUX_Dma1Trigout0ToAoi1InTrigger
enumerator kINPUTMUX_Dma1Trigout1ToAoi1InTrigger
enumerator kINPUTMUX_Dma1Trigout2ToAoi1InTrigger
AOI External trigger.
enumerator kINPUTMUX_PinInt0ToAoiExtTrigger

enumerator kINPUTMUX_PinInt1ToAoiExtTrigger
enumerator kINPUTMUX_Adc0IrqToAoiExtTrigger
enumerator kINPUTMUX_Adc1IrqToAoiExtTrigger
enumerator kINPUTMUX_Adc0Tcomp0ToAoiExtTrigger
enumerator kINPUTMUX_Adc1Tcomp0ToAoiExtTrigger
enumerator kINPUTMUX_Pwm0Sm0MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Pwm0Sm1MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Pwm0Sm2MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Pwm0Sm3MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Pwm1Sm0MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Pwm1Sm1MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Pwm1Sm2MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Pwm1Sm3MuxTrig01ToAoiExtTrigger
enumerator kINPUTMUX_Enc0CmpPosMatchToAoiExtTrigger
enumerator kINPUTMUX_Enc1CmpPosMatchToAoiExtTrigger
enumerator kINPUTMUX_Aoi0Out0ToAoiExtTrigger
enumerator kINPUTMUX_Aoi0Out1ToAoiExtTrigger
enumerator kINPUTMUX_Aoi0Out2ToAoiExtTrigger
enumerator kINPUTMUX_Aoi0Out3ToAoiExtTrigger
enumerator kINPUTMUX_Aoi1Out0ToAoiExtTrigger
enumerator kINPUTMUX_Aoi1Out1ToAoiExtTrigger
enumerator kINPUTMUX_Aoi1Out2ToAoiExtTrigger
enumerator kINPUTMUX_Aoi1Out3ToAoiExtTrigger
enumerator kINPUTMUX_TmprOutToAoiExtTrigger
HSCMP1 trigger.
enumerator kINPUTMUX_PinInt0ToHscmp1Trigger
enumerator kINPUTMUX_PinInt7ToHscmp1Trigger
enumerator kINPUTMUX_SctOut4ToHscmp1Trigger
enumerator kINPUTMUX_SctOut5ToHscmp1Trigger
enumerator kINPUTMUX_SctOut7ToHscmp1Trigger
enumerator kINPUTMUX_Ctimer0M3ToHscmp1Trigger
enumerator kINPUTMUX_Ctimer1M3ToHscmp1Trigger
enumerator kINPUTMUX_Ctimer2M3ToHscmp1Trigger

enumerator kINPUTMUX__Ctimer0M1ToHscmp1Trigger
enumerator kINPUTMUX__Ctimer4M1ToHscmp1Trigger
enumerator kINPUTMUX__ArmTxevToHscmp1Trigger
enumerator kINPUTMUX__GpioIntBmatchToHscmp1Trigger
enumerator kINPUTMUX__Adc0Tcomp1ToHscmp1Trigger
enumerator kINPUTMUX__Adc1Tcomp1ToHscmp1Trigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToHscmp1Trigger
enumerator kINPUTMUX__Enc0CmpPosMatchToHscmp1Trigger
enumerator kINPUTMUX__Enc1CmpPosMatchToHscmp1Trigger
enumerator kINPUTMUX__Aoi0Out0ToHscmp1Trigger
enumerator kINPUTMUX__Aoi0Out1ToHscmp1Trigger
enumerator kINPUTMUX__Aoi0Out2ToHscmp1Trigger
enumerator kINPUTMUX__Aoi0Out3ToHscmp1Trigger
enumerator kINPUTMUX__Aoi1Out0ToHscmp1Trigger
enumerator kINPUTMUX__Aoi1Out1ToHscmp1Trigger
enumerator kINPUTMUX__Aoi1Out2ToHscmp1Trigger
enumerator kINPUTMUX__Aoi1Out3ToHscmp1Trigger
enumerator kINPUTMUX__Dma0Trigout0ToHscmp1Trigger
enumerator kINPUTMUX__Dma0Trigout1ToHscmp1Trigger
enumerator kINPUTMUX__Dma0Trigout2ToHscmp1Trigger
HSCMP2 trigger.
enumerator kINPUTMUX__PinInt0ToHscmp2Trigger
enumerator kINPUTMUX__PinInt4ToHscmp2Trigger
enumerator kINPUTMUX__SctOut4ToHscmp2Trigger
enumerator kINPUTMUX__SctOut5ToHscmp2Trigger
enumerator kINPUTMUX__SctOut8ToHscmp2Trigger

enumerator kINPUTMUX__Ctimer0M3ToHscmp2Trigger
enumerator kINPUTMUX__Ctimer1M3ToHscmp2Trigger
enumerator kINPUTMUX__Ctimer2M3ToHscmp2Trigger
enumerator kINPUTMUX__Ctimer0M2ToHscmp2Trigger
enumerator kINPUTMUX__Ctimer4M2ToHscmp2Trigger
enumerator kINPUTMUX__ArmTxevToHscmp2Trigger
enumerator kINPUTMUX__GpiontBmatchToHscmp2Trigger
enumerator kINPUTMUX__Adc0Tcomp2ToHscmp2Trigger
enumerator kINPUTMUX__Adc1Tcomp2ToHscmp2Trigger
enumerator kINPUTMUX__Pwm0Sm0MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Pwm0Sm1MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Pwm0Sm2MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Pwm0Sm3MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Pwm1Sm0MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Pwm1Sm1MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Pwm1Sm2MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Pwm1Sm3MuxTrig01ToHscmp2Trigger
enumerator kINPUTMUX__Enc0CmpPosMatchToHscmp2Trigger
enumerator kINPUTMUX__Enc1CmpPosMatchToHscmp2Trigger
enumerator kINPUTMUX__Aoi0Out0ToHscmp2Trigger
enumerator kINPUTMUX__Aoi0Out1ToHscmp2Trigger
enumerator kINPUTMUX__Aoi0Out2ToHscmp2Trigger
enumerator kINPUTMUX__Aoi0Out3ToHscmp2Trigger
enumerator kINPUTMUX__Aoi1Out0ToHscmp2Trigger
enumerator kINPUTMUX__Aoi1Out1ToHscmp2Trigger
enumerator kINPUTMUX__Aoi1Out2ToHscmp2Trigger
enumerator kINPUTMUX__Aoi1Out3ToHscmp2Trigger
enumerator kINPUTMUX__Dma0Trigout0ToHscmp2Trigger
enumerator kINPUTMUX__Dma0Trigout1ToHscmp2Trigger
enumerator kINPUTMUX__Dma0Trigout2ToHscmp2Trigger

enum _inputmux_signal_t

INPUTMUX signal enable/disable type.

Values:

enumerator kINPUTMUX_FlexSpiRxToDmac0Ch0RequestEna
DMA0 REQ(DMA0_REQEN0) signal.

enumerator kINPUTMUX_FlexSpiTxToDmac0Ch1RequestEna

enumerator kINPUTMUX_PinInt0ToDmac0Ch2RequestEna

enumerator kINPUTMUX_PinInt1ToDmac0Ch3RequestEna

enumerator kINPUTMUX_PinInt2ToDmac0Ch4RequestEna

enumerator kINPUTMUX_PinInt3ToDmac0Ch5RequestEna

enumerator kINPUTMUX_Ctimer0M0ToDmac0Ch6RequestEna

enumerator kINPUTMUX_Ctimer0M1ToDmac0Ch7RequestEna

enumerator kINPUTMUX_Ctimer1M0ToDmac0Ch8RequestEna

enumerator kINPUTMUX_Ctimer1M1ToDmac0Ch9RequestEna

enumerator kINPUTMUX_Ctimer2M0ToDmac0Ch10RequestEna

enumerator kINPUTMUX_Ctimer2M1ToDmac0Ch11RequestEna

enumerator kINPUTMUX_Ctimer3M0ToDmac0Ch12RequestEna

enumerator kINPUTMUX_Ctimer3M1ToDmac0Ch13RequestEna

enumerator kINPUTMUX_Ctimer4M0ToDmac0Ch14RequestEna

enumerator kINPUTMUX_Ctimer4M1ToDmac0Ch15RequestEna

enumerator kINPUTMUX_CompOutToDmac0Ch16RequestEna

enumerator kINPUTMUX_OtrigAToDmac0Ch17RequestEna

enumerator kINPUTMUX_OtrigBToDmac0Ch18RequestEna

enumerator kINPUTMUX_OtrigCToDmac0Ch19RequestEna

enumerator kINPUTMUX_OtrigDToDmac0Ch20RequestEna

enumerator kINPUTMUX_SctDma0ToDmac0Ch21RequestEna

enumerator kINPUTMUX_SctDma1ToDmac0Ch22RequestEna

enumerator kINPUTMUX_Adc0Tcomp0ToDmac23Ch0RequestEna

enumerator kINPUTMUX_Adc1Tcomp0ToDmac24Ch0RequestEna

enumerator kINPUTMUX_Hscmp0ToDmac0Ch25RequestEna

enumerator kINPUTMUX_Hscmp1ToDmac0Ch26RequestEna

enumerator kINPUTMUX_Hscmp2ToDmac0Ch27RequestEna

enumerator kINPUTMUX_Aoi0Out0ToDmac0Ch28RequestEna

enumerator kINPUTMUX_Aoi0Out1ToDmac0Ch29RequestEna

enumerator kINPUTMUX_Aoi0Out2ToDmac0Ch30RequestEna

enumerator kINPUTMUX_Aoi0Out3ToDmac0Ch31RequestEna
DMA0 REQ(DMA0_REQEN0) signal.

enumerator kINPUTMUX__Aoi1Out0ToDmac0Ch32RequestEna
enumerator kINPUTMUX__Aoi1Out1ToDmac0Ch33RequestEna
enumerator kINPUTMUX__Aoi1Out2ToDmac0Ch34RequestEna
enumerator kINPUTMUX__Aoi1Out3ToDmac0Ch35RequestEna
enumerator kINPUTMUX__FlexPwm0ReqCapt0ToDmac0Ch36RequestEna
enumerator kINPUTMUX__FlexPwm0ReqCapt1ToDmac0Ch37RequestEna
enumerator kINPUTMUX__FlexPwm0ReqCapt2ToDmac0Ch38RequestEna
enumerator kINPUTMUX__FlexPwm0ReqCapt3ToDmac0Ch39RequestEna
enumerator kINPUTMUX__FlexPwm0ReqVal0ToDmac0Ch40RequestEna
enumerator kINPUTMUX__FlexPwm0ReqVal1ToDmac0Ch41RequestEna
enumerator kINPUTMUX__FlexPwm0ReqVal2ToDmac0Ch42RequestEna
enumerator kINPUTMUX__FlexPwm0ReqVal3ToDmac0Ch43RequestEna
enumerator kINPUTMUX__FlexPwm1ReqCapt0ToDmac0Ch44RequestEna
enumerator kINPUTMUX__FlexPwm1ReqCapt1ToDmac0Ch45RequestEna
enumerator kINPUTMUX__FlexPwm1ReqCapt2ToDmac0Ch46RequestEna
enumerator kINPUTMUX__FlexPwm1ReqCapt3ToDmac0Ch47RequestEna
enumerator kINPUTMUX__FlexPwm1ReqVal0ToDmac0Ch48RequestEna
enumerator kINPUTMUX__FlexPwm1ReqVal1ToDmac0Ch49RequestEna
enumerator kINPUTMUX__FlexPwm1ReqVal2ToDmac0Ch50RequestEna
enumerator kINPUTMUX__FlexPwm1ReqVal3ToDmac0Ch51RequestEna
enumerator kINPUTMUX__TmprOutToDmac0Ch52RequestEna
DMA1 REQ(DMA1_REQEN) signal.
enumerator kINPUTMUX__HsLspiRxToDmac1Ch2RequestEna
enumerator kINPUTMUX__HsLspiTxToDmac1Ch3RequestEna
enumerator kINPUTMUX__Flexcom0RxToDmac1Ch4RequestEna
enumerator kINPUTMUX__Flexcom0TxToDmac1Ch5RequestEna
enumerator kINPUTMUX__Flexcom1RxToDmac1Ch6RequestEna
enumerator kINPUTMUX__Flexcom1TxToDmac1Ch7RequestEna
enumerator kINPUTMUX__Flexcom3RxToDmac1Ch8RequestEna
enumerator kINPUTMUX__Flexcom3TxToDmac1Ch9RequestEna
enumerator kINPUTMUX__Dmic0Ch0ToDmac1Ch10RequestEna
enumerator kINPUTMUX__Dmic0Ch1ToDmac1Ch11RequestEna
enumerator kINPUTMUX__I3c0RxToDmac1Ch12RequestEna

enumerator kINPUTMUX_I3c0TxToDmac1Ch13RequestEna
DMA0 input trigger(DMA0_ITRIGEN0) source enable.

enumerator kINPUTMUX_Dmac0InputTriggerFlexSpiRxEna

enumerator kINPUTMUX_Dmac0InputTriggerFlexSpiTxEna

enumerator kINPUTMUX_Dmac0InputTriggerPinInt0Ena

enumerator kINPUTMUX_Dmac0InputTriggerPinInt1Ena

enumerator kINPUTMUX_Dmac0InputTriggerPinInt2Ena

enumerator kINPUTMUX_Dmac0InputTriggerPinInt3Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer0M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer0M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer1M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer1M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer2M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer2M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer3M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer3M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer4M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer4M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCompOutEna

enumerator kINPUTMUX_Dmac0InputTriggerOtrigAEna

enumerator kINPUTMUX_Dmac0InputTriggerOtrigBEna

enumerator kINPUTMUX_Dmac0InputTriggerOtrigCEna

enumerator kINPUTMUX_Dmac0InputTriggerOtrigDEna

enumerator kINPUTMUX_Dmac0InputTriggerSctDma0Ena

enumerator kINPUTMUX_Dmac0InputTriggerSctDma1Ena

enumerator kINPUTMUX_Dmac0InputTriggerAdc0Tcomp0Ena

enumerator kINPUTMUX_Dmac0InputTriggerAdc1Tcomp0Ena

enumerator kINPUTMUX_Dmac0InputTriggerHscmp0Ena

enumerator kINPUTMUX_Dmac0InputTriggerHscmp1Ena

enumerator kINPUTMUX_Dmac0InputTriggerHscmp2Ena

enumerator kINPUTMUX_Dmac0InputTriggerAoi0Out0Ena

enumerator kINPUTMUX_Dmac0InputTriggerAoi0Out1Ena

enumerator kINPUTMUX_Dmac0InputTriggerAoi0Out2Ena

```

enumerator kINPUTMUX__Dmac0InputTriggerAoi0Out3Ena
    DMA0 input trigger(DMA0_ITRIGEN1) source enable.
enumerator kINPUTMUX__Dmac0InputTriggerAoi1Out0Ena
enumerator kINPUTMUX__Dmac0InputTriggerAoi1Out1Ena
enumerator kINPUTMUX__Dmac0InputTriggerAoi1Out2Ena
enumerator kINPUTMUX__Dmac0InputTriggerAoi1Out3Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqCapt0Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqCapt1Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqCapt2Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqCapt3Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqVal0Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqVal1Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqVal2Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm0ReqVal3Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqCapt0Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqCapt1Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqCapt2Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqCapt3Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqVal0Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqVal1Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqVal2Ena
enumerator kINPUTMUX__Dmac0InputTriggerFlexPwm1ReqVal3Ena
enumerator kINPUTMUX__Dmac0InputTriggerTmprOutEna
    DMA1 input trigger(DMA1_ITRIGEN) source enable.
enumerator kINPUTMUX__Dmac1InputTriggerHsLspiRxEna
enumerator kINPUTMUX__Dmac1InputTriggerHsLspiTxEna
enumerator kINPUTMUX__Dmac1InputTriggerFlexcom0RxEna
enumerator kINPUTMUX__Dmac1InputTriggerFlexcom0TxEna
enumerator kINPUTMUX__Dmac1InputTriggerFlexcom1RxEna
enumerator kINPUTMUX__Dmac1InputTriggerFlexcom1TxEna
enumerator kINPUTMUX__Dmac1InputTriggerFlexcom3RxEna
enumerator kINPUTMUX__Dmac1InputTriggerFlexcom3TxEna
enumerator kINPUTMUX__Dmac1InputTriggerDmic0Ch0Ena
enumerator kINPUTMUX__Dmac1InputTriggerDmic0Ch1Ena

```

```
enumerator kINPUTMUX_Dmac1InputTriggerI3c0RxEna
enumerator kINPUTMUX_Dmac1InputTriggerI3c0TxEna
typedef enum _inputmux_connection_t inputmux_connection_t
    INPUTMUX connections type.
typedef enum _inputmux_signal_t inputmux_signal_t
    INPUTMUX signal enable/disable type.
SCT0_INMUX0
    Periphinmux IDs.
TIMER0CAPTSEL0
TIMER0TRIGIN
TIMER1CAPTSEL0
TIMER1TRIGIN
TIMER2CAPTSEL0
TIMER2TRIGIN
PINTSEL_PMUX_ID
PINTSEL0
DMA0_ITRIG_INMUX0
DMA0_OTRIG_INMUX0
FREQMEAS_REF_REG
FREQMEAS_TARGET_REG
TIMER3CAPTSEL0
TIMER3TRIGIN
TIMER4CAPTSEL0
TIMER4TRIGIN
PINTSECSEL0
DMA1_ITRIG_INMUX0
DMA1_OTRIG_INMUX0
HSCMP0_TRIGIN
ADC0_TRIG0
ADC1_TRIG0
DAC0_TRIGIN
DAC1_TRIGIN
DAC2_TRIGIN
ENC0TRIG
```

ENC0HOME
ENC0INDEX
ENC0PHASEB
ENC0PHASEA
ENC1TRIG
ENC1HOME
ENC1INDEX
ENC1PHASEB
ENC1PHASEA
PWM0_EXTSYNC0
PWM0_EXTA0
PWM0_EXTFORCETRIG
PWM0_FAULT0
PWM1_EXTSYNC0
PWM1_EXTA0
PWM1_EXTFORCETRIG
PWM1_FAULT0
PWM0_EXTCLKTRIG
PWM1_EXTCLKTRIG
AOI0_IN0
AOI1_IN0
AOI_EXT_TRIG0
HSCMP1_TRIGIN
HSCMP2_TRIGIN
DMA0_ITRIG_INMUX_32
DMA0_REQ_EN0_ID
DMA0_REQ_EN1_ID
DMA1_REQ_EN_ID
DMA0_ITRIG_EN0_ID
DMA0_ITRIG_EN1_ID
DMA1_ITRIG_EN_ID
ENA_SHIFT
PMUX_SHIFT

FSL_INPUTMUX_DRIVER_VERSION

Group interrupt driver version for SDK.

void INPUTMUX_Init(void *base)

Initialize INPUTMUX peripheral.

This function enables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

Return values

None. –

void INPUTMUX_AttachSignal(void *base, uint16_t index, *inputmux_connection_t* connection)

Attaches a signal.

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
```

In this example, INTMUX has 8 registers for PINT, PINT_SEL0~PINT_SEL7. With parameter index specified as 2, this function configures register PINT_SEL2.

Parameters

- base – Base address of the INPUTMUX peripheral.
- index – The serial number of destination register in the group of INPUTMUX registers with same name.
- connection – Applies signal from source signals collection to target signal.

Return values

None. –

void INPUTMUX_EnableSignal(void *base, *inputmux_signal_t* signal, bool enable)

Enable/disable a signal.

This function gates the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.
- signal – Enable signal register id and bit offset.
- enable – Selects enable or disable.

Return values

None. –

void INPUTMUX_Deinit(void *base)

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

Return values

None. –

2.50 Inline Prince Encryption Decryption

FSL_IPED_DRIVER_VERSION

IPED driver version. Version 2.2.0.

Current version: 2.2.0

Change log:

- Version 2.2.0
 - Renamed CSS to ELS
- Version 2.1.1
 - Fix build error due to renamed symbols
- Version 2.1.0
 - Add IPED_Config() (including CMPA write) and IPED_Reconfig() features.
- Version 2.0.0
 - Initial version

enum __iped_status

Values:

enumerator kStatus_IPED_RegionIsLocked

enum __iped_lock

Values:

enumerator kIPED_RegionUnlock

enumerator kIPED_RegionLock

enum __iped_cmpa

Values:

enumerator kIPED_SkipCMPA

enumerator kIPED_WriteCMPA

typedef uint32_t iped_region_t

typedef uint32_t iped_prince_rounds_t

typedef enum __iped_lock iped_lock_t

typedef enum __iped_cmpa iped_cmpa_t

typedef struct __flexspi_iped_region_option flexspi_iped_prot_region_option_t

typedef struct __flexspi_iped_region_arg flexspi_iped_region_arg_t

static inline void IPED_EncryptEnable(FLEXSPI_Type *base)

Enable data encryption.

This function enables IPED on-the-fly data encryption.

Parameters

- base – IPED peripheral address.

static inline void IPED__EncryptDisable(FLEXSPI_Type *base)

Disable data encryption.

This function disables IPED on-the-fly data encryption.

Parameters

- base – IPED peripheral address.

static inline void IPED__SetLock(FLEXSPI_Type *base, *iped_region_t* region)

Locks access for specified region registers or data mask register.

This function sets lock on specified region.

Parameters

- base – IPED peripheral address.
- region – number to lock

status_t IPED__SetRegionAddressRange(FLEXSPI_Type *base, *iped_region_t* region, uint32_t start_address, uint32_t end_address)

Sets IPED region address range.

This function configures IPED region address range.

Parameters

- base – IPED peripheral address.
- region – Selection of the IPED region to be configured.
- start_address – Start address for region.
- end_address – End address for region.

status_t IPED__GetRegionAddressRange(FLEXSPI_Type *base, *iped_region_t* region, uint32_t *start_address, uint32_t *end_address)

Gets IPED region base address.

This function reads current start and end address settings for selected region.

Parameters

- base – IPED peripheral address.
- region – Selection of the IPED region to be configured.
- start_address – Start address for region.
- end_address – End address for region.

status_t IPED__SetRegionIV(FLEXSPI_Type *base, *iped_region_t* region, const uint8_t iv[8])

Sets the IPED region IV.

This function sets specified AES IV for the given region.

Parameters

- base – IPED peripheral address.
- region – Selection of the IPED region to be configured.
- iv – 64-bit AES IV in little-endian byte order.

static inline void IPED__SetPrinceRounds(FLEXSPI_Type *base, *iped_prince_rounds_t* rounds)

Sets the IPED region IV.

This function sets specified AES IV for the given region.

Parameters

- base – IPED peripheral address.

- rounds – Number of PRINCE rounds used during encryption/decryption

status_t IPED_Configure(*api_core_context_t* *coreCtx, *flexspi_iped_region_arg_t* *config, *iped_lock_t* lock, *iped_cmpa_t* writeCmpa)

Configures IPED setting.

This function does the initial IPED configuration via ROM IAP API call. IPED_SR_x configuration for each region configuration is stored into FFR (CMPA). IPED IV erase counters (MCTR_INT_IV_CTRx) in CFPA are updated accordingly.

Note: This function is expected to be called once in the device lifetime, typically during the initial device provisioning (especially if programming the CMPA pages in PFR flash is enabled).

Parameters

- coreCtx – The pointer to the ROM API driver context structure.
- config – The pointer to the IPED driver configuration structure.
- lock – Locks the IPED configuration, if CMPA write enabled, also sets the IPEDx_START bits[1:0] 01 - Enabled, 10,11 - Enabled & locked
- writeCmpa – If selected, IPED configuration will be programmed in PFR flash using ROM API. Note: This can not be reverted!!

Return values

- kStatus_Success –
- kStatus_CommandUnsupported –
- kStatus_InvalidArgument –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatusMemoryRangeInvalid –
- kStatus_Fail –
- kStatus_OutOfRange –
- kStatus_SPI_BaudrateNotSupport –

status_t IPED_Reconfigure(*api_core_context_t* *coreCtx, *flexspi_iped_region_arg_t* *config)

Configures IPED setting.

This function is used to re-configure IPED IP based on configuration stored in FFR. This function also needs to be called after wake up from power-down mode to regenerate IV encryption key in ELS key store whose presence is necessary for correct IPED operation during erase and write operations to encrypted regions of internal flash memory (dependency for correct operation of MEM_Erase() and MEM_Write() after wake up from power-down mode).

Parameters

- coreCtx – The pointer to the ROM API driver context structure.
- config – The pointer to the IPED driver configuration structure. If NULL CMPA configuration is read and used. Note: when providing config structure, you have to call Reconfigure for each IPED region individually starting with Region 0. Region 0 must be enabled as a base region.

Return values

- kStatus_Success –
- kStatus_Fail –

kIPED_Region0

IPED region 0

kIPED_Region1

IPED region 1

kIPED_Region2

IPED region 2

kIPED_Region3

IPED region 3

kIPED_PrinceRounds12

kIPED_PrinceRounds22

IPED_TAG

IPED fixed tag in flexspi_iped_region_arg_t structure.

IPED_TAG_SHIFT

IPED_REGION_COUNT

IPED region count.

IPED_RW_ENABLE_VAL

IPED_RW_DISABLE_VAL

NXP_DIE_EXT_MEM_ENC_SK

Define for ELS key store indexes.

NXP_DIE_MEM_IV_ENC_SK

CFPA_VER_OFFSET

CFPA version and IV indexes (see Protected Flash Region table)

CFPA_IPED_IV_OFFSET

CFPA_SCRATCH_VER

CFPA scratch version and IV addresses (see Protected Flash Region table)

CFPA_SCRATCH_IV

IPED_START_ADDR_LOCK_EN_MASK

CMPA start address, end address, lock and enable bit-field masks (see Protected Flash Region table)

IPED_START_ADDR_MASK

IPED_END_ADDR_MASK

SYSCON_ELS_KDF_MASK

KDF mask and key properties for NXP_DIE_MEM_IV_ENC_SK (see SYSCON documentation)

CMPA_IPED_START_OFFSET

CMPA Start address index (see Protected Flash Region table)

IPED_ADDRESS_MASK

CMPA Start address mask (see Protected Flash Region table)

IPED_ENABLE_MASK

CMPA Start address enable/lock mask bits (see Protected Flash Region table)

CMPA_PAGE_SIZE

 CMPA page size (see Protected Flash Region table)

struct __flexspi_iped_region_option

 #include <fsl_iped.h>

struct __flexspi_iped_region_arg

 #include <fsl_iped.h>

struct IPED_CMPA_page

 #include <fsl_iped.h> IPED - CMPA page layout.

Public Members

uint8_t RESERVED_0[144]

 Reserved 0, offset: 0x00

__IO uint32_t IPED0_START

 IPED0_START, offset: 0x90

__IO uint32_t IPED0_END

 IPED0_END, offset: 0x94

__IO uint32_t IPED1_START

 IPED1_START, offset: 0x98

__IO uint32_t IPED1_END

 IPED1_END, offset: 0x9C

__IO uint32_t IPED2_START

 IPED2_START, offset: 0xA0

__IO uint32_t IPED2_END

 IPED2_END, offset: 0xA4

__IO uint32_t IPED3_START

 IPED3_START, offset: 0xA8

__IO uint32_t IPED3_END

 IPED3_END, offset: 0xAC

uint8_t RESERVED_1[336]

 Reserved 1, offset: 0xB0

2.51 IRTC: IRTC Driver

status_t IRTC_Init(RTC_Type *base, const irtc_config_t *config)

 Ungates the IRTC clock and configures the peripheral for basic operation.

 This function initiates a soft-reset of the IRTC module, this has not effect on DST, calendar-ing, standby time and tamper detect registers.

Note: This API should be called at the beginning of the application using the IRTC driver.

Parameters

- base – IRTC peripheral base address

- `config` – Pointer to user's IRTC config structure.

Returns

`kStatus_Success` If the driver is initialized successfully.

Returns

`kStatus_Fail` if we cannot disable register write protection

Returns

`kStatus_InvalidArgument` If the input parameters are wrong.

`status_t` IRTC_Deinit(RTC_Type *base)

Gate the IRTC clock.

Parameters

- `base` – IRTC peripheral base address

Returns

`kStatus_Success` If the driver is initialized successfully.

Returns

`kStatus_InvalidArgument` If the input parameters are wrong.

`void` IRTC_GetDefaultConfig(*irtc_config_t* *config)

Fill in the IRTC config struct with the default settings.

The default values are:

```
config->wakeupSelect = true;
config->timerStdMask = false;
config->alarmMatch = kRTC_MatchSecMinHr;
```

Parameters

- `config` – Pointer to user's IRTC config structure.

`status_t` IRTC_SetDatetime(RTC_Type *base, const *irtc_datetime_t* *datetime)

Sets the IRTC date and time according to the given time structure.

The IRTC counter is started after the time is set.

Parameters

- `base` – IRTC peripheral base address
- `datetime` – Pointer to structure where the date and time details to set are stored

Returns

`kStatus_Success`: success in setting the time and starting the IRTC
`kStatus_InvalidArgument`: failure. An error occurs because the datetime format is incorrect.

`void` IRTC_GetDatetime(RTC_Type *base, *irtc_datetime_t* *datetime)

Gets the IRTC time and stores it in the given time structure.

Parameters

- `base` – IRTC peripheral base address
- `datetime` – Pointer to structure where the date and time details are stored.

`status_t` IRTC_SetAlarm(RTC_Type *base, const *irtc_datetime_t* *alarmTime)

Sets the IRTC alarm time.

Note: `weekDay` field of `alarmTime` is not used during alarm match and should be set to 0

Parameters

- `base` – RTC peripheral base address
- `alarmTime` – Pointer to structure where the alarm time is stored.

Returns

`kStatus_Success`: success in setting the alarm
`kStatus_InvalidArgument`: error in setting the alarm. Error occurs because the alarm datetime format is incorrect.

```
void IRTC_GetAlarm(RTC_Type *base, irtc_datetime_t *datetime)
```

Returns the IRTC alarm time.

Parameters

- `base` – RTC peripheral base address
- `datetime` – Pointer to structure where the alarm date and time details are stored.

```
static inline void IRTC_EnableInterrupts(RTC_Type *base, uint32_t mask)
```

Enables the selected IRTC interrupts.

Parameters

- `base` – IRTC peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `irtc_interrupt_enable_t`

```
static inline void IRTC_DisableInterrupts(RTC_Type *base, uint32_t mask)
```

Disables the selected IRTC interrupts.

Parameters

- `base` – IRTC peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `irtc_interrupt_enable_t`

```
static inline uint32_t IRTC_GetEnabledInterrupts(RTC_Type *base)
```

Gets the enabled IRTC interrupts.

Parameters

- `base` – IRTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `irtc_interrupt_enable_t`

```
static inline uint32_t IRTC_GetStatusFlags(RTC_Type *base)
```

Gets the IRTC status flags.

Parameters

- `base` – IRTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `irtc_status_flags_t`

```
static inline void IRTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)
```

Clears the IRTC status flags.

Parameters

- `base` – IRTC peripheral base address

- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `irtc_status_flags_t`

`void IRTC_SetDaylightTime(RTC_Type *base, const irtc_daylight_time_t *datetime)`

Sets the IRTC daylight savings start and stop date and time.

It also enables the daylight saving bit in the IRTC control register

Parameters

- `base` – IRTC peripheral base address
- `datetime` – Pointer to a structure where the date and time details are stored.

`void IRTC_GetDaylightTime(RTC_Type *base, irtc_daylight_time_t *datetime)`

Gets the IRTC daylight savings time and stores it in the given time structure.

Parameters

- `base` – IRTC peripheral base address
- `datetime` – Pointer to a structure where the date and time details are stored.

`void IRTC_SetCoarseCompensation(RTC_Type *base, uint8_t compensationValue, uint8_t compensationInterval)`

Enables the coarse compensation and sets the value in the IRTC compensation register.

Parameters

- `base` – IRTC peripheral base address
- `compensationValue` – Compensation value is a 2's complement value.
- `compensationInterval` – Compensation interval.

`void IRTC_SetFineCompensation(RTC_Type *base, uint8_t integralValue, uint8_t fractionValue, bool accumulateFractional)`

Enables the fine compensation and sets the value in the IRTC compensation register.

Parameters

- `base` – The IRTC peripheral base address
- `integralValue` – Compensation integral value; twos complement value of the integer part
- `fractionValue` – Compensation fraction value expressed as number of clock cycles of a fixed 4.194304Mhz clock that have to be added.
- `accumulateFractional` – Flag indicating if we want to add to previous fractional part; true: Add to previously accumulated fractional part, false: Start afresh and overwrite current value

`static inline void IRTC_EnableSubsecondCounter(RTC_Type *base, bool enable)`

Enable the RTC wake-up timer.

1HZ clock out selected via call to API `IRTC_ConfigClockOut` in order for the subsecond counter to synchronize with the `RTC_SECONDS` counter.

Parameters

- `base` – RTC peripheral base address
- `enable` – Use/Un-use the sub-second counter.
 - true: Use RTC wake-up timer at the same time.
 - false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.

```
static inline uint32_t IRTC_GetSubsecondCount(RTC_Type *base)
```

Read the actual RTC sub-second COUNT value.

Parameters

- base – RTC peripheral base address

Returns

The actual RTC sub-second COUNT value.

```
static inline void IRTC_SetWakeupCount(RTC_Type *base, bool enable1kHzClk, uint32_t  
wakeUpValue)
```

Set countdown value to the RTC wake timer counter register.

Parameters

- base – RTC peripheral base address
- enable1kHzClk – Enable 1kHz clock source for the wake timer, else use the 32kHz clock.
- wakeUpValue – The value to be loaded into the WAKE register in wake timer counter.

```
static inline uint32_t IRTC_GetWakeupCount(RTC_Type *base)
```

Read the actual value from the WAKE register value in RTC wake timer.

Parameters

- base – RTC peripheral base address

Returns

The actual value of the WAKE register value in wake timer counter.

```
FSL_IRTC_DRIVER_VERSION
```

```
enum __irtc_clock_select
```

IRTC clock select.

Values:

```
enumerator kIRTC_Clk16K
```

16.384 kHz clock is selected.

```
enumerator kIRTC_Clk32K
```

32.768 kHz clock is selected.

```
enum __irtc_interrupt_enable
```

List of IRTC interrupts.

Values:

```
enumerator kIRTC_AlarmInterruptEnable
```

Alarm Interrupt Enable

```
enumerator kIRTC_DayInterruptEnable
```

Days Interrupt Enable

```
enumerator kIRTC_HourInterruptEnable
```

Hours Interrupt Enable

```
enumerator kIRTC_MinInterruptEnable
```

Minutes Interrupt Enable

```
enumerator kIRTC_1hzInterruptEnable
```

1 Hz interval Interrupt Enable

enumerator kIRTC_2hzInterruptEnable
2 Hz interval Interrupt Enable

enumerator kIRTC_4hzInterruptEnable
4 Hz interval Interrupt Enable

enumerator kIRTC_8hzInterruptEnable
8 Hz interval Interrupt Enable

enumerator kIRTC_16hzInterruptEnable
16 Hz interval Interrupt Enable

enumerator kIRTC_32hzInterruptEnable
32 Hz interval Interrupt Enable

enumerator kIRTC_64hzInterruptEnable
64 Hz interval Interrupt Enable

enumerator kIRTC_128hzInterruptEnable
128 Hz interval Interrupt Enable

enumerator kIRTC_256hzInterruptEnable
256 Hz interval Interrupt Enable

enumerator kIRTC_512hzInterruptEnable
512 Hz interval Interrupt Enable

enumerator kIRTC_WakeTimerInterruptEnable
Wake timer Interrupt Enable

enumerator kIRTC_TamperQueueFullInterruptEnable
Tamper queue full Interrupt Enable

enum _irtc_status_flags

List of IRTC flags.

Values:

enumerator kIRTC_AlarmFlag
Alarm Status flag

enumerator kIRTC_DayFlag
Days Status flag

enumerator kIRTC_HourFlag
Hour Status flag

enumerator kIRTC_MinFlag
Minutes Status flag

enumerator kIRTC_1hzFlag
1 Hz interval status flag

enumerator kIRTC_2hzFlag
2 Hz interval status flag

enumerator kIRTC_4hzFlag
4 Hz interval status flag

enumerator kIRTC_8hzFlag
8 Hz interval status flag

enumerator kIRTC_16hzFlag
16 Hz interval status flag

enumerator kIRTC_32hzFlag
32 Hz interval status flag

enumerator kIRTC_64hzFlag
64 Hz interval status flag

enumerator kIRTC_128hzFlag
128 Hz interval status flag

enumerator kIRTC_256hzFlag
256 Hz interval status flag

enumerator kIRTC_512hzFlag
512 Hz interval status flag

enumerator kIRTC_InvalidFlag
Indicates if time/date counters are invalid

enumerator kIRTC_WriteProtFlag
Write protect enable status flag

enumerator kIRTC_CmpIntFlag
Compensation interval status flag

enumerator kIRTC_CmpDoneFlag
Compensation done flag

enumerator kIRTC_BusErrFlag
Bus error flag

enumerator kIRTC_WakeTimerFlag
Wake timer status flag

enum _irtc_alarm_match
IRTC alarm match options.

Values:

enumerator kRTC_MatchSecMinHr
Only match second, minute and hour

enumerator kRTC_MatchSecMinHrDay
Only match second, minute, hour and day

enumerator kRTC_MatchSecMinHrDayMnth
Only match second, minute, hour, day and month

enumerator kRTC_MatchSecMinHrDayMnthYr
Only match second, minute, hour, day, month and year

enum _irtc_clockout_sel
IRTC clockout select.

Values:

enumerator kIRTC_ClkoutNo
No clock out

enumerator kIRTC_ClkoutFine1Hz
clock out fine 1Hz

enumerator kIRTC_Clkout32kHz

clock out 32.768kHz

enumerator kIRTC_ClkoutCoarse1Hz

clock out coarse 1Hz

typedef enum *_irtc_clock_select* irtc_clock_select_t

IRTC clock select.

typedef enum *_irtc_interrupt_enable* irtc_interrupt_enable_t

List of IRTC interrupts.

typedef enum *_irtc_status_flags* irtc_status_flags_t

List of IRTC flags.

typedef enum *_irtc_alarm_match* irtc_alarm_match_t

IRTC alarm match options.

typedef enum *_irtc_clockout_sel* irtc_clockout_sel_t

IRTC clockout select.

typedef struct *_irtc_datetime* irtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_irtc_daylight_time* irtc_daylight_time_t

Structure is used to hold the daylight saving time.

typedef struct *_irtc_config* irtc_config_t

RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the `IRTC_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 IRTC_SetWriteProtection(RTC_Type *base, bool lock)

Locks or unlocks IRTC registers for write access.

Note: When the registers are unlocked, they remain in unlocked state for 2 seconds, after which they are locked automatically. After power-on-reset, the registers come out unlocked and they are locked automatically 15 seconds after power on.

Parameters

- base – IRTC peripheral base address
- lock – true: Lock IRTC registers; false: Unlock IRTC registers.

Returns

kStatus_Success: if lock or unlock operation is successful
kStatus_Fail: if lock or unlock operation fails even after multiple retry attempts

static inline void IRTC_Reset(RTC_Type *base)

Performs a software reset on the IRTC module.

Clears contents of alarm, interrupt (status and enable except tamper interrupt enable bit) registers, STATUS[`CMP_DONE`] and STATUS[`BUS_ERR`]. This has no effect on DST, calendar-ing, standby time and tamper detect registers.

Parameters

- base – IRTC peripheral base address

```
void IRTC__ConfigClockOut(RTC_Type *base, irtc_clockout_sel_t clkOut)
```

Select which clock to output from RTC.

Select which clock to output from RTC for other modules to use inside SoC, for example, RTC subsystem needs RTC to output 1HZ clock for sub-second counter.

Parameters

- base – IRTC peripheral base address
- clkOut – select clock to use for output,

```
void IRTC__ConfigClockSelect(RTC_Type *base, irtc_clock_select_t clkSelect)
```

Select which clock is used by RTC.

Select which clock is used by RTC to output to the peripheral and divided to generate a 512 Hz clock and a 1 Hz clock.

Parameters

- base – IRTC peripheral base address
- clkSelect – select clock used by RTC

```
static inline void IRTC__EnableClockOutputToPeripheral(RTC_Type *base, bool enable)
```

Determines whether the selected clock is output to other peripherals.

Determines whether the selected clock is output to other peripherals.

Parameters

- base – IRTC peripheral base address
- enable – determine whether the selected clock is output to other peripherals

```
IRTC_STATUS_W1C_BITS
```

```
struct __irtc_datetime
```

#include <fsl_irtc.h> Structure is used to hold the date and time.

Public Members

```
uint16_t year
```

Range from 1984 to 2239.

```
uint8_t month
```

Range from 1 to 12.

```
uint8_t day
```

Range from 1 to 31 (depending on month).

```
uint8_t weekDay
```

Range from 0(Sunday) to 6(Saturday).

```
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 __irtc_daylight_time
```

#include <fsl_irtc.h> Structure is used to hold the daylight saving time.

Public Members

uint8_t startMonth

Range from 1 to 12

uint8_t endMonth

Range from 1 to 12

uint8_t startDay

Range from 1 to 31 (depending on month)

uint8_t endDay

Range from 1 to 31 (depending on month)

uint8_t startHour

Range from 0 to 23

uint8_t endHour

Range from 0 to 23

struct _irtc_config

#include <fsl_irtc.h> RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the IRTC_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

irtc_alarm_match_t alarmMatch

Pick one option from enumeration :: irtc_alarm_match_t

irtc_clock_select_t clockSelect

Pick one option from enumeration :: irtc_clock_select_t

bool disableClockOutput

true: The selected clock is not output to other peripherals; false: The selected clock is output to other peripherals

2.52 Intrusion and Tamper Response Controller

2.53 ITRC

status_t ITRC_SetActionToEvent(ITRC_Type *base, irtc_out_signals_t out, irtc_input_signals_t in, irtc_lock_t lock, irtc_enable_t enable)

Set ITRC Action to Event.

This function sets input Event signal to corresponding output Action response signal.

Parameters

- base – ITRC peripheral base address
- out – ITRC OUT signal action
- in – ITRC IN signal event

- lock – if set locks INx_SEL configuration. This can be cleared only by PMC Core reset.
- enable – if set input Event will be selected for output Action, otherwise disable (if not already locked).

Returns

kStatus_Success if success, kStatus_InvalidArgument otherwise

void ITRC_SetSWEvent0(ITRC_Type *base)

Trigger ITRC SW Event 0.

This function set SW_EVENT0 register with value !=0 which triggers ITRC SW Event 0.

Parameters

- base – ITRC peripheral base address

void ITRC_SetSWEvent1(ITRC_Type *base)

Trigger ITRC SW Event 1.

This function set SW_EVENT1 register with value !=0 which triggers ITRC SW Event 1.

Parameters

- base – ITRC peripheral base address

uint32_t ITRC_GetStatus(ITRC_Type *base)

Get ITRC Status.

This function returns ITRC register status.

Parameters

- base – ITRC peripheral base address

Returns

Value of ITRC STATUS register

status_t ITRC_ClearStatus(ITRC_Type *base, uint32_t word)

Clear ITRC status.

This function clears corresponding ITRC event or action in STATUS register.

Parameters

- base – ITRC peripheral base address
- word – 32bit word represent corresponding event/action in STATUS register to be cleared (see ITRC_STATUS_INx/OUTx_STATUS)

Returns

kStatus_Success if success, kStatus_InvalidArgument otherwise

status_t ITRC_ClearAllStatus(ITRC_Type *base)

Clear All ITRC status.

This function clears all event and action status.

Parameters

- base – ITRC peripheral base address

Returns

kStatus_Success if success

status_t ITRC_Init(ITRC_Type *base)

Initialize ITRC.

This function initializes ITRC by enabling IRQ.

Parameters

- `base` – ITRC peripheral base address
- `conf` – ITRC configuration structure

Returns

Status of the init operation

`void ITRC_Deinit(ITRC_Type *base)`

Deinitialize ITRC.

This function deinitializes ITRC by disabling IRQ.

Parameters

- `base` – ITRC peripheral base address

`FSL_ITRC_DRIVER_VERSION`

Defines ITRC driver version 2.4.0.

Change log:

- Version 2.4.0
 - Rework the input signal definition for better flexibility
- Version 2.3.0
 - Update names of `kITRC_SwEvent1/2` to `kITRC_SwEvent0/1` to align with RM
- Version 2.2.0
 - Update driver to new version and input events
- Version 2.1.0
 - Make SYSCON glitch platform dependent
- Version 2.0.0
 - initial version

`enum _itrc_input_signals`

Values:

`enum _itrc_lock`

Values:

enumerator `kITRC_Unlock`

enumerator `kITRC_Lock`

`enum _itrc_enable`

Values:

enumerator `kITRC_Enable`

enumerator `kITRC_Disable`

`enum _itrc_out_signals`

Values:

enumerator `kITRC_Irq`

enumerator `kITRC_ElsReset`

enumerator `kITRC_PufZeroize`

enumerator `kITRC_RamZeroize`

```
    enumerator kITRC_ChipReset
    enumerator kITRC_TamperOut
    enumerator kITRC_TamperOut1
typedef enum _itrc_input_signals itrc_input_signals_t
typedef enum _itrc_lock itrc_lock_t
typedef enum _itrc_enable itrc_enable_t
typedef enum _itrc_out_signals itrc_out_signals_t
void ITRC0_DriverIRQHandler(void)
ITRC_STATUS_IN2_STATUS_MASK
ITRC_STATUS_IN3_STATUS_MASK
ITRC_STATUS_IN9_STATUS_MASK
ITRC_STATUS1_IN17_STATUS_MASK
ITRC_STATUS1_IN19_STATUS_MASK
ITRC_STATUS1_IN24_21_STATUS_MASK
ITRC_STATUS1_IN32_25_STATUS_MASK
ITRC_STATUS1_IN46_STATUS_MASK
IN_0_15_EVENTS_MASK
OUT_ACTIONS_MASK
ITRC_OUT_COUNT
ITRC
```

2.54 Common Driver

```
FSL_COMMON_DRIVER_VERSION
    common driver version.
DEBUG_CONSOLE_DEVICE_TYPE_NONE
    No debug console.
DEBUG_CONSOLE_DEVICE_TYPE_UART
    Debug console based on UART.
DEBUG_CONSOLE_DEVICE_TYPE_LPUART
    Debug console based on LPUART.
DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
    Debug console based on LPSCI.
DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
    Debug console based on USBCDC.
DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
    Debug console based on FLEXCOMM.
```


DEBUG_CONSOLE_DEVICE_TYPE_IUART

Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART

Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART

Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO

Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI

Debug console based on QSCI.

MIN(*a*, *b*)

Computes the minimum of *a* and *b*.

MAX(*a*, *b*)

Computes the maximum of *a* and *b*.

UINT16_MAX

Max value of uint16_t type.

UINT32_MAX

Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(addr, val)

Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)

Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)

Set the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)

Clear the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specified by *clearBits* and set the bits specified by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true**, else return **false**.

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

AT_NONCACHEABLE_SECTION(var)

Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)

Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_CACHE_LINE_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

AT_QUICKACCESS_SECTION_CODE(func)

Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)

Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)

Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

RAMFUNCTION_SECTION_CODE(func)

Place function in ram.

enum __status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI
Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI
Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART
Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C
Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C
Group number for LPI2C status codes.

enumerator kStatusGroup_UART
Group number for UART status codes.

enumerator kStatusGroup_I2C
Group number for UART status codes.

enumerator kStatusGroup_LPSCI
Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART
Group number for LPUART status codes.

enumerator kStatusGroup_SPI
Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator `kStatusGroup_FLEXCOMM_I2C`
Group number for FLEXCOMM I2C status codes

enumerator `kStatusGroup_I2S`
Group number for I2S status codes

enumerator `kStatusGroup_IUART`
Group number for IUART status codes

enumerator `kStatusGroup_CSI`
Group number for CSI status codes

enumerator `kStatusGroup_MIPIDSI`
Group number for MIPI DSI status codes

enumerator `kStatusGroup_SDRAMC`
Group number for SDRAMC status codes.

enumerator `kStatusGroup_POWER`
Group number for POWER status codes.

enumerator `kStatusGroup_ENET`
Group number for ENET status codes.

enumerator `kStatusGroup_PHY`
Group number for PHY status codes.

enumerator `kStatusGroup_TRGMUX`
Group number for TRGMUX status codes.

enumerator `kStatusGroup_SMARTCARD`
Group number for SMARTCARD status codes.

enumerator `kStatusGroup_LMEM`
Group number for LMEM status codes.

enumerator `kStatusGroup_QSPI`
Group number for QSPI status codes.

enumerator `kStatusGroup_DMA`
Group number for DMA status codes.

enumerator `kStatusGroup_EDMA`
Group number for EDMA status codes.

enumerator `kStatusGroup_DMAMGR`
Group number for DMAMGR status codes.

enumerator `kStatusGroup_FLEXCAN`
Group number for FlexCAN status codes.

enumerator `kStatusGroup_LTC`
Group number for LTC status codes.

enumerator `kStatusGroup_FLEXIO_CAMERA`
Group number for FLEXIO CAMERA status codes.

enumerator `kStatusGroup_LPC_SPI`
Group number for LPC_SPI status codes.

enumerator `kStatusGroup_LPC_USART`
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPi
Group number for ECSPi status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator `kStatusGroup_I3C`
Group number for I3C status codes

enumerator `kStatusGroup_LPC_I2C_1`
Group number for LPC_I2C_1 status codes.

enumerator `kStatusGroup_NOTIFIER`
Group number for NOTIFIER status codes.

enumerator `kStatusGroup_DebugConsole`
Group number for debug console status codes.

enumerator `kStatusGroup_SEMC`
Group number for SEMC status codes.

enumerator `kStatusGroup_ApplicationRangeStart`
Starting number for application groups.

enumerator `kStatusGroup_IAP`
Group number for IAP status codes

enumerator `kStatusGroup_SFA`
Group number for SFA status codes

enumerator `kStatusGroup_SPC`
Group number for SPC status codes.

enumerator `kStatusGroup_PUF`
Group number for PUF status codes.

enumerator `kStatusGroup_TOUCH_PANEL`
Group number for touch panel status codes

enumerator `kStatusGroup_VBAT`
Group number for VBAT status codes

enumerator `kStatusGroup_XSPI`
Group number for XSPI status codes

enumerator `kStatusGroup_PNGDEC`
Group number for PNGDEC status codes

enumerator `kStatusGroup_JPEGDEC`
Group number for JPEGDEC status codes

enumerator `kStatusGroup_AUDMIX`
Group number for AUDMIX status codes

enumerator `kStatusGroup_HAL_GPIO`
Group number for HAL GPIO status codes.

enumerator `kStatusGroup_HAL_UART`
Group number for HAL UART status codes.

enumerator `kStatusGroup_HAL_TIMER`
Group number for HAL TIMER status codes.

enumerator `kStatusGroup_HAL_SPI`
Group number for HAL SPI status codes.

enumerator `kStatusGroup_HAL_I2C`
Group number for HAL I2C status codes.

enumerator `kStatusGroup_HAL_FLASH`
Group number for HAL FLASH status codes.

enumerator `kStatusGroup_HAL_PWM`
Group number for HAL PWM status codes.

enumerator `kStatusGroup_HAL_RNG`
Group number for HAL RNG status codes.

enumerator `kStatusGroup_HAL_I2S`
Group number for HAL I2S status codes.

enumerator `kStatusGroup_HAL_ADC_SENSOR`
Group number for HAL ADC SENSOR status codes.

enumerator `kStatusGroup_TIMERMANAGER`
Group number for TiMER MANAGER status codes.

enumerator `kStatusGroup_SERIALMANAGER`
Group number for SERIAL MANAGER status codes.

enumerator `kStatusGroup_LED`
Group number for LED status codes.

enumerator `kStatusGroup_BUTTON`
Group number for BUTTON status codes.

enumerator `kStatusGroup_EXTERN_EEPROM`
Group number for EXTERN EEPROM status codes.

enumerator `kStatusGroup_SHELL`
Group number for SHELL status codes.

enumerator `kStatusGroup_MEM_MANAGER`
Group number for MEM MANAGER status codes.

enumerator `kStatusGroup_LIST`
Group number for List status codes.

enumerator `kStatusGroup_OSA`
Group number for OSA status codes.

enumerator `kStatusGroup_COMMON_TASK`
Group number for Common task status codes.

enumerator `kStatusGroup_MSG`
Group number for messaging status codes.

enumerator `kStatusGroup_SDK_OCOTP`
Group number for OCOTP status codes.

enumerator `kStatusGroup_SDK_FLEXSPINOR`
Group number for FLEXSPINOR status codes.

enumerator `kStatusGroup_CODEC`
Group number for codec status codes.

enumerator `kStatusGroup_ASRC`
Group number for codec status ASRC.

enumerator `kStatusGroup_OTFAD`
Group number for codec status codes.

enumerator `kStatusGroup_SDIO_SLV`
Group number for SDIO_SLV status codes.

enumerator `kStatusGroup_MECC`
Group number for MECC status codes.

enumerator `kStatusGroup_ENET_QOS`
Group number for ENET_QOS status codes.

enumerator `kStatusGroup_LOG`
Group number for LOG status codes.

enumerator `kStatusGroup_I3CBUS`
Group number for I3CBUS status codes.

enumerator `kStatusGroup_QSCI`
Group number for QSCI status codes.

enumerator `kStatusGroup_ELEMU`
Group number for ELEMU status codes.

enumerator `kStatusGroup_QUEUEDSPI`
Group number for QSPI status codes.

enumerator `kStatusGroup_POWER_MANAGER`
Group number for POWER_MANAGER status codes.

enumerator `kStatusGroup_IPED`
Group number for IPED status codes.

enumerator `kStatusGroup_ELS_PKC`
Group number for ELS PKC status codes.

enumerator `kStatusGroup_CSS_PKC`
Group number for CSS PKC status codes.

enumerator `kStatusGroup_HOSTIF`
Group number for HOSTIF status codes.

enumerator `kStatusGroup_CLIF`
Group number for CLIF status codes.

enumerator `kStatusGroup_BMA`
Group number for BMA status codes.

enumerator `kStatusGroup_NETC`
Group number for NETC status codes.

enumerator `kStatusGroup_ELE`
Group number for ELE status codes.

enumerator `kStatusGroup_GLIKEY`
Group number for GLIKEY status codes.

enumerator `kStatusGroup_AON_POWER`
Group number for AON_POWER status codes.

enumerator `kStatusGroup_AON_COMMON`
Group number for AON_COMMON status codes.

enumerator `kStatusGroup_ENDAT3`
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
Generic status for Success.

enumerator kStatus_Fail
Generic status for Fail.

enumerator kStatus_ReadOnly
Generic status for read only failure.

enumerator kStatus_OutOfRange
Generic status for out of range access.

enumerator kStatus_InvalidArgument
Generic status for invalid argument check.

enumerator kStatus_Timeout
Generic status for timeout.

enumerator kStatus_NoTransferInProgress
Generic status for no transfer in progress.

enumerator kStatus_Busy
Generic status for module is busy.

enumerator kStatus_NoData
Generic status for no data is found for the operation.

typedef int32_t status_t
Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values

The – allocated memory.

`void SDK_Free(void *ptr)`

Free memory.

Parameters

- `ptr` – The memory to be release.

`void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)`

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- `delayTime_us` – Delay time in unit of microsecond.
- `coreClock_Hz` – Core clock frequency with Hz.

`static inline status_t EnableIRQ(IRQn_Type interrupt)`

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- `interrupt` – The IRQ number.

Return values

- `kStatus_Success` – Interrupt enabled successfully
- `kStatus_Fail` – Failed to enable the interrupt

`static inline status_t DisableIRQ(IRQn_Type interrupt)`

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- `interrupt` – The IRQ number.

Return values

- `kStatus_Success` – Interrupt disabled successfully
- `kStatus_Fail` – Failed to disable the interrupt

`static inline status_t EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)`

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

Returns

Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management

mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

void EnableDeepSleepIRQ(IRQn_Type interrupt)

Enable specific interrupt for wake-up from deep-sleep mode.

Enable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note: This function also enables the interrupt in the NVIC (EnableIRQ() is called internally).

Parameters

- interrupt – The IRQ number.

void DisableDeepSleepIRQ(IRQn_Type interrupt)

Disable specific interrupt for wake-up from deep-sleep mode.

Disable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note: This function also disables the interrupt in the NVIC (DisableIRQ() is called internally).

Parameters

- interrupt – The IRQ number.

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.

2.55 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 `TPRCTRL` 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 `TPRCTRL` 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.

ADC_OFSTRIM_OFSTRIM_MAX**ADC_OFSTRIM_OFSTRIM_SIGN****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;
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.

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;
```

(continues on next page)

(continued from previous page)

```

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_SetOffsetValue(ADC_Type *base, int16_t value)

Set trim value for offset.

Note: For 16-bit conversions, each increment is 1/2 LSB resulting in a programmable offset range of -256 LSB to 255.5 LSB; For 12-bit conversions, each increment is 1/32 LSB resulting in a programmable offset range of -16 LSB to 15.96875 LSB.

Parameters

- base – LPADC peripheral base address.
- value – Offset trim value, is a 10-bit signed value between -512 and 511.

static inline void LPADC_GetOffsetValue(ADC_Type *base, int16_t *pValue)

Get trim value of offset.

Parameters

- base – LPADC peripheral base address.
- pValue – Pointer to the variable in type of int16_t to store offset value.

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.

2.56 Lpc_freqme

void FREQME_Init(FREQME_Type *base, const *freq_measure_config_t* *config)

Initialize freqme module, set operate mode, operate mode attribute and initialize measurement cycle.

Parameters

- base – FREQME peripheral base address.
- config – The pointer to module basic configuration, please refer to *freq_measure_config_t*.

void FREQME_GetDefaultConfig(*freq_measure_config_t* *config)

Get default configuration.

```
config->operateMode = kFREQME_FreqMeasurementMode;
config->operateModeAttribute.refClkScaleFactor = 0U;
config->enableContinuousMode = false;
config->startMeasurement = false;
```


Parameters

- `config` – The pointer to module basic configuration, please refer to `freq_measure_config_t`.

```
static inline void FREQME_StartMeasurementCycle(FREQME_Type *base)
```

Start frequency or pulse width measurement process.

Parameters

- `base` – FREQME peripheral base address.

```
static inline void FREQME_TerminateMeasurementCycle(FREQME_Type *base)
```

Force the termination of any measurement cycle currently in progress and resets RESULT or just reset RESULT if the module in idle state.

Parameters

- `base` – FREQME peripheral base address.

```
static inline void FREQME_EnableContinuousMode(FREQME_Type *base, bool enable)
```

Enable/disable Continuous mode.

Parameters

- `base` – FREQME peripheral base address.
- `enable` – Used to enable/disable continuous mode,
 - **true** Enable Continuous mode.
 - **false** Disable Continuous mode.

```
static inline bool FREQME_CheckContinuousMode(FREQME_Type *base)
```

Check whether continuous mode is enabled.

Parameters

- `base` – FREQME peripheral base address.

Return values

- **True** – Continuous mode is enabled, the measurement is performed continuously.
- **False** – Continuous mode is disabled.

```
static inline void FREQME_SetOperateMode(FREQME_Type *base, freqme_operate_mode_t  
                                         operateMode)
```

Set operate mode of freqme module.

Parameters

- `base` – FREQME peripheral base address.
- `operateMode` – The operate mode to be set, please refer to `freqme_operate_mode_t`.

```
static inline bool FREQME_CheckOperateMode(FREQME_Type *base)
```

Check module's operate mode.

Parameters

- `base` – FREQME peripheral base address.

Return values

- **True** – Pulse width measurement mode.
- **False** – Frequency measurement mode.

static inline void FREQME_SetMinExpectedValue(FREQME_Type *base, uint32_t minValue)

Set the minimum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- minValue – The minimum value to set, please note that this value is 31 bits width.

static inline void FREQME_SetMaxExpectedValue(FREQME_Type *base, uint32_t maxValue)

Set the maximum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- maxValue – The maximum value to set, please note that this value is 31 bits width.

uint32_t FREQME_CalculateTargetClkFreq(FREQME_Type *base, uint32_t refClkFrequency)

Calculate the frequency of selected target clock

Note: The formula: $F_{target} = (RESULT - 2) * F_{reference} / 2^{REF\_SCALE}$.

Note: This function only useful when the operate mode is selected as frequency measurement mode.

Parameters

- base – FREQME peripheral base address.
- refClkFrequency – The frequency of reference clock.

Returns

The frequency of target clock the unit is Hz, if the output result is 0, please check the module's operate mode.

static inline uint8_t FREQME_GetReferenceClkScaleValue(FREQME_Type *base)

Get reference clock scaling factor.

Parameters

- base – FREQME peripheral base address.

Returns

Reference clock scaling factor, the reference count cycle is 2^{ref_scale} .

static inline void FREQME_SetPulsePolarity(FREQME_Type *base, freqme_pulse_polarity_t pulsePolarity)

Set pulse polarity when operate mode is selected as Pulse Width Measurement mode.

Parameters

- base – FREQME peripheral base address.
- pulsePolarity – The pulse polarity to be set, please refer to freqme_pulse_polarity_t.

static inline bool FREQME_CheckPulsePolarity(FREQME_Type *base)

Check pulse polarity when the operate mode is selected as pulse width measurement mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Low period.
- False – High period.

static inline uint32_t `FREQME_GetMeasurementResult(FREQME_Type *base)`

Get measurement result, if operate mode is selected as pulse width measurement mode this function can be used to calculate pulse width.

Note: Pulse width = counter result / Frequency of target clock.

Parameters

- base – FREQME peripheral base address.

Returns

Measurement result.

static inline uint32_t `FREQME_GetInterruptStatusFlags(FREQME_Type *base)`

Get interrupt status flags, such as overflow interrupt status flag, underflow interrupt status flag, and so on.

Parameters

- base – FREQME peripheral base address.

Returns

Current interrupt status flags, should be the OR'ed value of `_freqme_interrupt_status_flags`.

static inline void `FREQME_ClearInterruptStatusFlags(FREQME_Type *base, uint32_t statusFlags)`

Clear interrupt status flags.

Parameters

- base – FREQME peripheral base address.
- statusFlags – The combination of interrupt status flags to clear, should be the OR'ed value of `_freqme_interrupt_status_flags`.

static inline void `FREQME_EnableInterrupts(FREQME_Type *base, uint32_t masks)`

Enable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to enable, should be the OR'ed value of `_freqme_interrupt_enable`.

static inline void `FREQME_DisableInterrupts(FREQME_Type *base, uint32_t masks)`

Disable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to disable, should be the OR'ed value of `_freqme_interrupt_enable`.

`FSL_FREQME_DRIVER_VERSION`

FREQME driver version 2.1.2.

enum `_freqme_interrupt_status_flags`

The enumeration of interrupt status flags. .

Values:

enumerator `kFREQME__UnderflowInterruptStatusFlag`

Indicate the measurement is just done and the result is less than minimum value.

enumerator `kFREQME__OverflowInterruptStatusFlag`

Indicate the measurement is just done and the result is greater than maximum value.

enumerator `kFREQME__ReadyInterruptStatusFlag`

Indicate the measurement is just done and the result is ready to read.

enumerator `kFREQME__AllInterruptStatusFlags`

All interrupt status flags.

enum `_freqme_interrupt_enable`

The enumeration of interrupts, including underflow interrupt, overflow interrupt, and result ready interrupt. .

Values:

enumerator `kFREQME__UnderflowInterruptEnable`

Enable interrupt when the result is less than minimum value.

enumerator `kFREQME__OverflowInterruptEnable`

Enable interrupt when the result is greater than maximum value.

enumerator `kFREQME__ReadyInterruptEnable`

Enable interrupt when a measurement completes and the result is ready.

enum `_freqme_operate_mode`

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

Values:

enumerator `kFREQME__FreqMeasurementMode`

The module works in the frequency measurement mode.

enumerator `kFREOME__PulseWidthMeasurementMode`

The module works in the pulse width measurement mode.

enum `_freqme_pulse_polarity`

The enumeration of pulse polarity.

Values:

enumerator `kFREQME__PulseHighPeriod`

Select high period of the reference clock.

enumerator `kFREQME__PulseLowPeriod`

Select low period of the reference clock.

typedef enum `_freqme_operate_mode` `freqme_operate_mode_t`

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

typedef enum `_freqme_pulse_polarity` `freqme_pulse_polarity_t`

The enumeration of pulse polarity.

```
typedef union _freqme_mode_attribute freqme_mode_attribute_t
```

The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

```
typedef struct _freq_measure_config freq_measure_config_t
```

The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

```
union _freqme_mode_attribute
```

#include <fsl_freqme.h> The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

Public Members

uint8_t refClkScaleFactor

Only useful in frequency measurement operate mode, used to set the reference clock counter scaling factor.

freqme_pulse_polarity_t pulsePolarity

Only Useful in pulse width measurement operate mode, used to set period polarity.

```
struct _freq_measure_config
```

#include <fsl_freqme.h> The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

Public Members

freqme_operate_mode_t operateMode

Select operate mode, please refer to *freqme_operate_mode_t*.

freqme_mode_attribute_t operateModeAttribute

Used to set the attribute of the selected operate mode, if the operate mode is selected as kFREQME_FreqMeasurementMode set *freqme_mode_attribute_t::refClkScaleFactor*, if operate mode is selected as kFREOME_PulseWidthMeasurementMode, please set *freqme_mode_attribute_t::pulsePolarity*.

bool enableContinuousMode

Enable/disable continuous mode, if continuous mode is enable, the measurement is performed continuously and the result for the last completed measurement is available in the result register.

2.57 GPIO: General Purpose I/O

```
void GPIO_PortInit(GPIO_Type *base, uint32_t port)
```

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

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

```
void GPIO_PinInit(GPIO_Type *base, uint32_t port, uint32_t pin, const gpio_pin_config_t *config)
```

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or output pin configuration:

```
Define a digital input pin configuration,  
gpio_pin_config_t config =  
{  
    kGPIO_DigitalInput,  
    0,  
}  
Define a digital output pin configuration,  
gpio_pin_config_t config =  
{  
    kGPIO_DigitalOutput,  
    0,  
}
```

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- pin – GPIO pin number
- config – GPIO pin configuration pointer

```
static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t port, uint32_t pin, uint8_t output)
```

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

Parameters

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

```
static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t port, uint32_t pin)
```

Reads the current input value of the GPIO PIN.

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.

- 1: corresponding pin input high-logic level.

FSL_GPIO_DRIVER_VERSION

LPC GPIO driver version.

FSL_GPIO_DRIVER_VERSION

LPC GPIO driver version.

enum _gpio_pin_direction

LPC GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput

Set current pin as digital input

enumerator kGPIO_DigitalOutput

Set current pin as digital output

enum _gpio_pin_direction

LPC 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_IntDisabled

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

LPC GPIO direction definition.

typedef struct _gpio_pin_config gpio_pin_config_t

The GPIO pin configuration structure.

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

typedef enum _gpio_pin_direction gpio_pin_direction_t

LPC 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
```

The GPIO pin configuration structure.

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

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

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

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

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

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

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

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

Reverses current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

```
void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const gpio_pin_config_t *Config)
```

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or output pin configuration:

```
// Define a digital input pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalInput,
    0,
}
//Define a digital output pin configuration,
gpio_pin_config_t config =
{
    kGPIO_DigitalOutput,
    0,
}
```

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- pin – GPIO pin number

- Config – GPIO pin configuration pointer

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

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number
- pinInterruptMode – GPIO interrupt mode

`struct __gpio_pin_config`

`#include <fsl_gpio.h>` The GPIO pin configuration structure.

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

Public Members

`gpio_pin_direction_t` pinDirection

GPIO direction, input or output

`uint8_t` outputLogic

Set default output logic, no use in input

Set a default output logic, which has no use in input

`gpio_pin_direction_t` direction

Specifies the pin direction.

2.58 GPIO: General Purpose I/O

2.59 IOCON: I/O pin configuration

`FSL_IOCON_DRIVER_VERSION`

IOCON driver version.

`typedef struct __iocon_group` iocon_group_t

Array of IOCON pin definitions passed to IOCON_SetPinMuxing() must be in this format.

`__STATIC_INLINE void IOCON_PinMuxSet (IOCON_Type *base, uint8_t port, uint8_t pin, uint32_t modefunc)`

Sets I/O Control pin mux.

Parameters

- base – : The base of IOCON peripheral on the chip
- port – : GPIO port to mux
- pin – : GPIO pin to mux
- modefunc – : OR'ed values of type IOCON_*

Returns

Nothing

```
__STATIC_INLINE void IOCON_SetPinMuxing (IOCON_Type *base,  
const iocon_group_t *pinArray, uint32_t arrayLength)
```

Set all I/O Control pin muxing.

Parameters

- base – : The base of IOCON peripheral on the chip
- pinArray – : Pointer to array of pin mux selections
- arrayLength – : Number of entries in pinArray

Returns

Nothing

FSL_COMPONENT_ID

IOCON_FUNC0

IOCON function and mode selection definitions.

Note: See the User Manual for specific modes and functions supported by the various pins.
Selects pin function 0

IOCON_FUNC1

Selects pin function 1

IOCON_FUNC2

Selects pin function 2

IOCON_FUNC3

Selects pin function 3

IOCON_FUNC4

Selects pin function 4

IOCON_FUNC5

Selects pin function 5

IOCON_FUNC6

Selects pin function 6

IOCON_FUNC7

Selects pin function 7

struct __iocon_group

#include <fsl_iocon.h> Array of IOCON pin definitions passed to IOCON_SetPinMuxing()
must be in this format.

2.60 MCAN: Controller Area Network Driver

```
void MCAN_Init(CAN_Type *base, const mcan_config_t *config, uint32_t sourceClock_Hz)
```

Initializes an MCAN instance.

This function initializes the MCAN module with user-defined settings. This example shows how to set up the mcan_config_t parameters and how to call the MCAN_Init function by passing in these parameters.

```
mcan_config_t config;
config->baudRateA = 500000U;
config->baudRateD = 1000000U;
config->enableCanfdNormal = false;
config->enableCanfdSwitch = false;
config->enableLoopBackInt = false;
config->enableLoopBackExt = false;
config->enableBusMon = false;
MCAN_Init(CANFD0, &config, 8000000UL);
```

Parameters

- base – MCAN peripheral base address.
- config – Pointer to the user-defined configuration structure.
- sourceClock_Hz – MCAN Protocol Engine clock source frequency in Hz.

void MCAN_Deinit(CAN_Type *base)

Deinitializes an MCAN instance.

This function deinitializes the MCAN module.

Parameters

- base – MCAN peripheral base address.

void MCAN_GetDefaultConfig(*mcan_config_t* *config)

Gets the default configuration structure.

This function initializes the MCAN configuration structure to default values. The default values are as follows. config->baudRateA = 500000U; config->baudRateD = 1000000U; config->enableCanfdNormal = false; config->enableCanfdSwitch = false; config->enableLoopBackInt = false; config->enableLoopBackExt = false; config->enableBusMon = false;

Parameters

- config – Pointer to the MCAN configuration structure.

static inline void MCAN_EnterInitialMode(CAN_Type *base)

MCAN enters initialization mode.

After enter initialization mode, users can write access to the protected configuration registers.

Parameters

- base – MCAN peripheral base address.

static inline void MCAN_EnterNormalMode(CAN_Type *base)

MCAN enters normal mode.

After initialization, INIT bit in CCCR register must be cleared to enter normal mode thus synchronizes to the CAN bus and ready for communication.

Parameters

- base – MCAN peripheral base address.

static inline void MCAN_SetMsgRAMBase(CAN_Type *base, uint32_t value)

Sets the MCAN Message RAM base address.

This function sets the Message RAM base address.

Parameters

- base – MCAN peripheral base address.

- value – Desired Message RAM base.

static inline uint32_t MCAN_GetMsgRAMBase(CAN_Type *base)

Gets the MCAN Message RAM base address.

This function gets the Message RAM base address.

Parameters

- base – MCAN peripheral base address.

Returns

Message RAM base address.

bool MCAN_CalculateImprovedTimingValues(uint32_t baudRate, uint32_t sourceClock_Hz, *mcan_timing_config_t* *pconfig)

Calculates the improved timing values by specific baudrates for classical CAN.

Parameters

- baudRate – The classical CAN speed in bps defined by user
- sourceClock_Hz – The Source clock data speed in bps. Zero to disable baudrate switching
- pconfig – Pointer to the MCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

bool MCAN_CalculateSpecifiedTimingValues(uint32_t sourceClock_Hz, *mcan_timing_config_t* *pconfig, const *mcan_timing_param_t* *pParamConfig)

Calculates the specified timing values for classical CAN with user-defined settings.

User can specify baudrates, sample point position, bus length, and transceiver propagation delay. This example shows how to set up the *mcan_timing_param_t* parameters and how to call the this function by passing in these parameters.

```
mcan_timing_config_t timing_config;
mcan_timing_param_t timing_param;
timing_param.busLength = 1U;
timing_param.propTxRx = 230U;
timing_param.nominalbaudRate = 500000U;
timing_param.nominalSP = 800U;
MCAN_CalculateSpecifiedTimingValues(MCAN_CLK_FREQ, &timing_config, &timing_param);
```

Note that due to integer division will sacrifice the precision, actual sample point may not equal to expected. If actual sample point is not in allowed 2% range, this function will return false. So it is better to select higher source clock when baudrate is relatively high. This will ensure more time quanta and higher precision of sample point. Parameter busLength and propTxRx are optional and intended to verify whether propagation delay is too long to corrupt sample point. User can set these parameter zero if you do not want to consider this factor.

Parameters

- sourceClock_Hz – The Source clock data speed in bps.
- pconfig – Pointer to the MCAN timing configuration structure.
- config – Pointer to the MCAN timing parameters structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

```
void MCAN_SetArbitrationTimingConfig(CAN_Type *base, const mcan_timing_config_t *config)
```

Sets the MCAN protocol arbitration phase timing characteristic.

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MCAN_Init() and fill the baud rate field with a desired value. This provides the default arbitration phase timing characteristics.

Note that calling MCAN_SetArbitrationTimingConfig() overrides the baud rate set in MCAN_Init().

Parameters

- base – MCAN peripheral base address.
- config – Pointer to the timing configuration structure.

```
status_t MCAN_SetBaudRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t
                           baudRate_Bps)
```

Set Baud Rate of MCAN classic mode.

This function set the baud rate of MCAN base on MCAN_CalculateImprovedTimingValues() API calculated timing values.

Parameters

- base – MCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- baudRate_Bps – Baud Rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only has Nominal phase) successfully.

```
bool MCAN_FDCalculateImprovedTimingValues(uint32_t baudRate, uint32_t baudRateFD,
                                           uint32_t sourceClock_Hz, mcan_timing_config_t
                                           *pconfig)
```

Calculates the improved timing values by specific baudrates for CANFD.

Parameters

- baudRate – The CANFD bus control speed in bps defined by user
- baudRateFD – The CANFD bus data speed in bps defined by user
- sourceClock_Hz – The Source clock data speed in bps.
- pconfig – Pointer to the MCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

```
bool MCAN_FDCalculateSpecifiedTimingValues(uint32_t sourceClock_Hz, mcan_timing_config_t
                                           *pconfig, const mcan_timing_param_t
                                           *pParamConfig)
```

Calculates the specified timing values for CANFD with user-defined settings.

User can specify baudrates, sample point position, bus length, and transceiver propagation delay. This example shows how to set up the *mcan_timing_param_t* parameters and how to call the this function by passing in these parameters.

```
mcan_timing_config_t timing_config;
mcan_timing_param_t timing_param;
timing_param.busLength = 1U;
timing_param.propTxRx = 230U;
timing_param.nominalbaudRate = 500000U;
timing_param.nominalSP = 800U;
```

(continues on next page)

(continued from previous page)

```

timing_param.databaudRate = 4000000U;
timing_param.dataSP = 700U;
MCAN_FDCalculateSpecifiedTimingValues(MCAN_CLK_FREQ, &timing_config, &timing_param);

```

Note that due to integer division will sacrifice the precision, actual sample point may not equal to expected. So it is better to select higher source clock when baudrate is relatively high. Select higher nominal baudrate when source clock is relatively high because large clock predivider will lead to less time quanta in data phase. This function will set predivider in arbitration phase equal to data phase. These methods will ensure more time quanta and higher precision of sample point. Parameter busLength and propTxRx are optional and intended to verify whether propagation delay is too long to corrupt sample point. User can set these parameter zero if you do not want to consider this factor.

Parameters

- sourceClock_Hz – The Source clock data speed in bps.
- pconfig – Pointer to the MCAN timing configuration structure.
- config – Pointer to the MCAN timing parameters structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

status_t MCAN_SetBaudRateFD(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t baudRateN_Bps, uint32_t baudRateD_Bps)

Set Baud Rate of MCAN FD mode.

This function set the baud rate of MCAN FD base on MCAN_FDCalculateImprovedTimingValues API calculated timing values.

Parameters

- base – MCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- baudRateN_Bps – Nominal Baud Rate in Bps.
- baudRateD_Bps – Data Baud Rate in Bps.

Returns

kStatus_Success - Set CAN FD baud rate (include Nominal and Data phase) successfully.

void MCAN_SetDataTimingConfig(CAN_Type *base, const *mcan_timing_config_t* *config)

Sets the MCAN protocol data phase timing characteristic.

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the MCAN_Init() and fill the baud rate field with a desired value. This provides the default data phase timing characteristics.

Note that calling MCAN_SetArbitrationTimingConfig() overrides the baud rate set in MCAN_Init().

Parameters

- base – MCAN peripheral base address.
- config – Pointer to the timing configuration structure.

void MCAN_SetRx Fifo0Config(CAN_Type *base, const *mcan_rx_fifo_config_t* *config)

Configures an MCAN receive fifo 0 buffer.

This function sets start address, element size, watermark, operation mode and datafield size of the receive fifo 0.

Parameters

- base – MCAN peripheral base address.
- config – The receive fifo 0 configuration structure.

void MCAN_SetRxFifo1Config(CAN_Type *base, const *mcan_rx_fifo_config_t* *config)

Configures an MCAN receive fifo 1 buffer.

This function sets start address, element size, watermark, operation mode and datafield size of the receive fifo 1.

Parameters

- base – MCAN peripheral base address.
- config – The receive fifo 1 configuration structure.

void MCAN_SetRxBufferConfig(CAN_Type *base, const *mcan_rx_buffer_config_t* *config)

Configures an MCAN receive buffer.

This function sets start address and datafield size of the receive buffer.

Parameters

- base – MCAN peripheral base address.
- config – The receive buffer configuration structure.

void MCAN_SetTxEventFifoConfig(CAN_Type *base, const *mcan_tx_fifo_config_t* *config)

Configures an MCAN transmit event fifo.

This function sets start address, element size, watermark of the transmit event fifo.

Parameters

- base – MCAN peripheral base address.
- config – The transmit event fifo configuration structure.

void MCAN_SetTxBufferConfig(CAN_Type *base, const *mcan_tx_buffer_config_t* *config)

Configures an MCAN transmit buffer.

This function sets start address, element size, fifo/queue mode and datafield size of the transmit buffer.

Parameters

- base – MCAN peripheral base address.
- config – The transmit buffer configuration structure.

void MCAN_SetFilterConfig(CAN_Type *base, const *mcan_frame_filter_config_t* *config)

Set filter configuration.

This function sets remote and non masking frames in global filter configuration, also the start address, list size in standard/extended ID filter configuration.

Parameters

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.

status_t MCAN_SetMessageRamConfig(CAN_Type *base, const *mcan_memory_config_t* *config)

Set Message RAM related configuration.

Note: This function include Standard/extended ID filter, Rx FIFO 0/1, Rx buffer, Tx event FIFO and Tx buffer configurations

Parameters

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.

Return values

- kStatus_Success – - Message RAM related configuration Successfully.
- kStatus_Fail – - Message RAM related configure fail due to wrong address parameter.

```
void MCAN_SetSTDFilterElement(CAN_Type *base, const mcan_frame_filter_config_t *config,  
                             const mcan_std_filter_element_config_t *filter, uint8_t idx)
```

Set standard message ID filter element configuration.

Parameters

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.
- filter – The MCAN standard message ID filter element configuration.
- idx – The standard message ID filter element index.

```
void MCAN_SetEXTFilterElement(CAN_Type *base, const mcan_frame_filter_config_t *config,  
                             const mcan_ext_filter_element_config_t *filter, uint8_t idx)
```

Set extended message ID filter element configuration.

Parameters

- base – MCAN peripheral base address.
- config – The MCAN filter configuration.
- filter – The MCAN extended message ID filter element configuration.
- idx – The extended message ID filter element index.

```
static inline uint32_t MCAN_GetStatusFlag(CAN_Type *base, uint32_t mask)
```

Gets the MCAN module interrupt flags.

This function gets all MCAN interrupt status flags.

Parameters

- base – MCAN peripheral base address.
- mask – The ORed MCAN interrupt mask.

Returns

MCAN status flags which are ORed.

```
static inline void MCAN_ClearStatusFlag(CAN_Type *base, uint32_t mask)
```

Clears the MCAN module interrupt flags.

This function clears MCAN interrupt status flags.

Parameters

- base – MCAN peripheral base address.
- mask – The ORed MCAN interrupt mask.

```
static inline bool MCAN_GetRxBufferStatusFlag(CAN_Type *base, uint8_t idx)
```

Gets the new data flag of specific Rx Buffer.

This function gets new data flag of specific Rx Buffer.

Parameters

- base – MCAN peripheral base address.
- idx – Rx Buffer index.

Returns

Rx Buffer new data status flag.

static inline void MCAN_ClearRxBufferStatusFlag(CAN_Type *base, uint8_t idx)

Clears the new data flag of specific Rx Buffer.

This function clears new data flag of specific Rx Buffer.

Parameters

- base – MCAN peripheral base address.
- idx – Rx Buffer index.

static inline void MCAN_EnableInterrupts(CAN_Type *base, uint32_t line, uint32_t mask)

Enables MCAN interrupts according to the provided interrupt line and mask.

This function enables the MCAN interrupts according to the provided interrupt line and mask. The mask is a logical OR of enumeration members.

Parameters

- base – MCAN peripheral base address.
- line – Interrupt line number, 0 or 1.
- mask – The interrupts to enable.

static inline void MCAN_EnableTransmitBufferInterrupts(CAN_Type *base, uint8_t idx)

Enables MCAN Tx Buffer interrupts according to the provided index.

This function enables the MCAN Tx Buffer interrupts.

Parameters

- base – MCAN peripheral base address.
- idx – Tx Buffer index.

static inline void MCAN_DisableTransmitBufferInterrupts(CAN_Type *base, uint8_t idx)

Disables MCAN Tx Buffer interrupts according to the provided index.

This function disables the MCAN Tx Buffer interrupts.

Parameters

- base – MCAN peripheral base address.
- idx – Tx Buffer index.

static inline void MCAN_DisableInterrupts(CAN_Type *base, uint32_t mask)

Disables MCAN interrupts according to the provided mask.

This function disables the MCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members.

Parameters

- base – MCAN peripheral base address.
- mask – The interrupts to disable.

uint32_t MCAN_IsTransmitRequestPending(CAN_Type *base, uint8_t idx)

Gets the Tx buffer request pending status.

This function returns Tx Message Buffer transmission request pending status.

Parameters

- base – MCAN peripheral base address.
- idx – The MCAN Tx Buffer index.

uint32_t MCAN_IsTransmitOccurred(CAN_Type *base, uint8_t idx)

Gets the Tx buffer transmission occurred status.

This function returns Tx Message Buffer transmission occurred status.

Parameters

- base – MCAN peripheral base address.
- idx – The MCAN Tx Buffer index.

status_t MCAN_WriteTxBuffer(CAN_Type *base, uint8_t idx, const *mcan_tx_buffer_frame_t* *pTxFrame)

Writes an MCAN Message to the Transmit 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 – MCAN peripheral base address.
- idx – The MCAN Tx Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

status_t MCAN_ReadRxBuffer(CAN_Type *base, uint8_t idx, *mcan_rx_buffer_frame_t* *pRxFrame)

Reads an MCAN Message from Rx Buffer.

This function reads a CAN message from the Rx Buffer in the Message RAM.

Parameters

- base – MCAN peripheral base address.
- idx – The MCAN Rx Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

kStatus_Success – Read Message from Rx Buffer successfully.

status_t MCAN_ReadRxFifo(CAN_Type *base, uint8_t fifoBlock, *mcan_rx_buffer_frame_t* *pRxFrame)

Reads an MCAN Message from Rx FIFO.

This function reads a CAN message from the Rx FIFO in the Message RAM.

Parameters

- base – MCAN peripheral base address.
- fifoBlock – Rx FIFO block 0 or 1.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

kStatus_Success – Read Message from Rx FIFO successfully.

static inline void MCAN_TransmitAddRequest(CAN_Type *base, uint8_t idx)

Tx Buffer add request to send message out.

This function add sending request to corresponding Tx Buffer.

Parameters

- base – MCAN peripheral base address.
- idx – Tx Buffer index.

static inline void MCAN__TransmitCancelRequest(CAN_Type *base, uint8_t idx)

Tx Buffer cancel sending request.

This function clears Tx buffer request pending bit.

Parameters

- base – MCAN peripheral base address.
- idx – Tx Buffer index.

status_t MCAN__TransferSendBlocking(CAN_Type *base, uint8_t idx, mcan_tx_buffer_frame_t *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

Parameters

- base – MCAN peripheral base pointer.
- idx – The MCAN 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 MCAN__TransferReceiveBlocking(CAN_Type *base, uint8_t idx, mcan_rx_buffer_frame_t *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

Parameters

- base – MCAN peripheral base pointer.
- idx – The MCAN buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Rx Message Buffer Successfully.
- kStatus_Fail – - No new message.

status_t MCAN__TransferReceiveFifoBlocking(CAN_Type *base, uint8_t fifoBlock, mcan_rx_buffer_frame_t *pRxFrame)

Performs a polling receive transaction from Rx FIFO on the CAN bus.

Note that a transfer handle does not need to be created before calling this API.

Parameters

- base – MCAN peripheral base pointer.
- fifoBlock – Rx FIFO block, 0 or 1.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.

- `kStatus_Fail` – - No new message in Rx FIFO.

```
void MCAN_TransferCreateHandle(CAN_Type *base, mcan_handle_t *handle,  
                             mcan_transfer_callback_t callback, void *userData)
```

Initializes the MCAN handle.

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

Parameters

- `base` – MCAN peripheral base address.
- `handle` – MCAN handle pointer.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

```
status_t MCAN_TransferSendNonBlocking(CAN_Type *base, mcan_handle_t *handle,  
                                     mcan_buffer_transfer_t *xfer)
```

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` – MCAN peripheral base address.
- `handle` – MCAN handle pointer.
- `xfer` – MCAN Buffer transfer structure. See the `mcan_buffer_transfer_t`.

Return values

- `kStatus_Success` – Start Tx Buffer sending process successfully.
- `kStatus_Fail` – Write Tx Buffer failed.
- `kStatus_MCAN_TxBusy` – Tx Buffer is in use.

```
status_t MCAN_TransferReceiveFifoNonBlocking(CAN_Type *base, uint8_t fifoBlock,  
                                             mcan_handle_t *handle, mcan_fifo_transfer_t  
                                             *xfer)
```

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` – MCAN peripheral base address.
- `handle` – MCAN handle pointer.
- `fifoBlock` – Rx FIFO block, 0 or 1.
- `xfer` – MCAN Rx FIFO transfer structure. See the `mcan_fifo_transfer_t`.

Return values

- `kStatus_Success` – - Start Rx FIFO receiving process successfully.
- `kStatus_MCAN_RxFifo0Busy` – - Rx FIFO 0 is currently in use.
- `kStatus_MCAN_RxFifo1Busy` – - Rx FIFO 1 is currently in use.

`void MCAN__TransferAbortSend(CAN_Type *base, mcan_handle_t *handle, uint8_t bufferIdx)`

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- `base` – MCAN peripheral base address.
- `handle` – MCAN handle pointer.
- `bufferIdx` – The MCAN Buffer index.

`void MCAN__TransferAbortReceiveFifo(CAN_Type *base, uint8_t fifoBlock, mcan_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` – MCAN peripheral base address.
- `fifoBlock` – MCAN Fifo block, 0 or 1.
- `handle` – MCAN handle pointer.

`void MCAN__TransferHandleIRQ(CAN_Type *base, mcan_handle_t *handle)`

MCAN IRQ handle function.

This function handles the MCAN Error, the Buffer, and the Rx FIFO IRQ request.

Parameters

- `base` – MCAN peripheral base address.
- `handle` – MCAN handle pointer.

`FSL_MCAN_DRIVER_VERSION`

MCAN driver version.

MCAN transfer status.

Values:

enumerator `kStatus_MCAN_TxBusy`

Tx Buffer is Busy.

enumerator `kStatus_MCAN_TxIdle`

Tx Buffer is Idle.

enumerator `kStatus_MCAN_RxBusy`

Rx Buffer is Busy.

enumerator `kStatus_MCAN_RxIdle`

Rx Buffer is Idle.

enumerator `kStatus_MCAN_RxFifo0New`

New message written to Rx FIFO 0.

enumerator `kStatus_MCAN_RxFifo0Idle`

Rx FIFO 0 is Idle.

enumerator `kStatus_MCAN_RxFifo0Watermark`

Rx FIFO 0 fill level reached watermark.

enumerator kStatus_MCAN_RxFifo0Full
Rx FIFO 0 full.

enumerator kStatus_MCAN_RxFifo0Lost
Rx FIFO 0 message lost.

enumerator kStatus_MCAN_RxFifo1New
New message written to Rx FIFO 1.

enumerator kStatus_MCAN_RxFifo1Idle
Rx FIFO 1 is Idle.

enumerator kStatus_MCAN_RxFifo1Watermark
Rx FIFO 1 fill level reached watermark.

enumerator kStatus_MCAN_RxFifo1Full
Rx FIFO 1 full.

enumerator kStatus_MCAN_RxFifo1Lost
Rx FIFO 1 message lost.

enumerator kStatus_MCAN_RxFifo0Busy
Rx FIFO 0 is busy.

enumerator kStatus_MCAN_RxFifo1Busy
Rx FIFO 1 is busy.

enumerator kStatus_MCAN_ErrorStatus
MCAN Module Error and Status.

enumerator kStatus_MCAN_UnHandled
UnHandled Interrupt asserted.

enum _mcan_flags

MCAN status flags.

This provides constants for the MCAN status flags for use in the MCAN functions. Note: The CPU read action clears MCAN_ErrorFlag, therefore user need to read MCAN_ErrorFlag and distinguish which error is occur using _mcan_error_flags enumerations.

Values:

enumerator kMCAN_AccesstoRsvdFlag
CAN Synchronization Status.

enumerator kMCAN_ProtocolErrDIntFlag
Tx Warning Interrupt Flag.

enumerator kMCAN_ProtocolErrAIntFlag
Rx Warning Interrupt Flag.

enumerator kMCAN_BusOffIntFlag
Tx Error Warning Status.

enumerator kMCAN_ErrorWarningIntFlag
Rx Error Warning Status.

enumerator kMCAN_ErrorPassiveIntFlag
Rx Error Warning Status.

enum `_mcan_rx_fifo_flags`

MCAN Rx FIFO status flags.

The MCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO.

Values:

enumerator `kMCAN_RxFifo0NewFlag`

Rx FIFO 0 new message flag.

enumerator `kMCAN_RxFifo0WatermarkFlag`

Rx FIFO 0 watermark reached flag.

enumerator `kMCAN_RxFifo0FullFlag`

Rx FIFO 0 full flag.

enumerator `kMCAN_RxFifo0LostFlag`

Rx FIFO 0 message lost flag.

enumerator `kMCAN_RxFifo1NewFlag`

Rx FIFO 1 new message flag.

enumerator `kMCAN_RxFifo1WatermarkFlag`

Rx FIFO 1 watermark reached flag.

enumerator `kMCAN_RxFifo1FullFlag`

Rx FIFO 1 full flag.

enumerator `kMCAN_RxFifo1LostFlag`

Rx FIFO 1 message lost flag.

enum `_mcan_tx_flags`

MCAN Tx status flags.

The MCAN Tx Status enumerations are used to determine the status of the Tx Buffer/Event FIFO.

Values:

enumerator `kMCAN_TxTransmitCompleteFlag`

Transmission completed flag.

enumerator `kMCAN_TxTransmitCancelFinishFlag`

Transmission cancellation finished flag.

enumerator `kMCAN_TxEventFifoLostFlag`

Tx Event FIFO element lost.

enumerator `kMCAN_TxEventFifoFullFlag`

Tx Event FIFO full.

enumerator `kMCAN_TxEventFifoWatermarkFlag`

Tx Event FIFO fill level reached watermark.

enumerator `kMCAN_TxEventFifoNewFlag`

Tx Handler wrote Tx Event FIFO element flag.

enumerator `kMCAN_TxEventFifoEmptyFlag`

Tx FIFO empty flag.

enum `_mcan_interrupt_enable`

MCAN interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the MCAN Module interrupt configurations.

Values:

enumerator kMCAN_BusOffInterruptEnable
Bus Off interrupt.

enumerator kMCAN_ErrorInterruptEnable
Error interrupt.

enumerator kMCAN_WarningInterruptEnable
Rx Warning interrupt.

enum __mcan_frame_idformat
MCAN frame format.

Values:

enumerator kMCAN_FrameIDStandard
Standard frame format attribute.

enumerator kMCAN_FrameIDExtend
Extend frame format attribute.

enum __mcan_frame_type
MCAN frame type.

Values:

enumerator kMCAN_FrameTypeData
Data frame type attribute.

enumerator kMCAN_FrameTypeRemote
Remote frame type attribute.

enum __mcan_bytes_in_datafield
MCAN frame datafield size.

Values:

enumerator kMCAN_8ByteDatafield
8 byte data field.

enumerator kMCAN_12ByteDatafield
12 byte data field.

enumerator kMCAN_16ByteDatafield
16 byte data field.

enumerator kMCAN_20ByteDatafield
20 byte data field.

enumerator kMCAN_24ByteDatafield
24 byte data field.

enumerator kMCAN_32ByteDatafield
32 byte data field.

enumerator kMCAN_48ByteDatafield
48 byte data field.

enumerator kMCAN_64ByteDatafield
64 byte data field.

enum __mcan_fifo_type
MCAN Rx FIFO block number.

Values:

enumerator kMCAN_Fifo0
CAN Rx FIFO 0.

enumerator kMCAN_Fifo1
CAN Rx FIFO 1.

enum __mcan_fifo_opmode_config
MCAN FIFO Operation Mode.

Values:

enumerator kMCAN_FifoBlocking
FIFO blocking mode.

enumerator kMCAN_FifoOverwrite
FIFO overwrite mode.

enum __mcan_txmode_config
MCAN Tx FIFO/Queue Mode.

Values:

enumerator kMCAN_txFifo
Tx FIFO operation.

enumerator kMCAN_txQueue
Tx Queue operation.

enum __mcan_remote_frame_config
MCAN remote frames treatment.

Values:

enumerator kMCAN_filterFrame
Filter remote frames.

enumerator kMCAN_rejectFrame
Reject all remote frames.

enum __mcan_nonmasking_frame_config
MCAN non-masking frames treatment.

Values:

enumerator kMCAN_acceptinFifo0
Accept non-masking frames in Rx FIFO 0.

enumerator kMCAN_acceptinFifo1
Accept non-masking frames in Rx FIFO 1.

enumerator kMCAN_reject0
Reject non-masking frames.

enumerator kMCAN_reject1
Reject non-masking frames.

enum __mcan_fec_config
MCAN Filter Element Configuration.

Values:

enumerator kMCAN_disable
Disable filter element.

enumerator `kMCAN_storeinFifo0`
Store in Rx FIFO 0 if filter matches.

enumerator `kMCAN_storeinFifo1`
Store in Rx FIFO 1 if filter matches.

enumerator `kMCAN_reject`
Reject ID if filter matches.

enumerator `kMCAN_setprio`
Set priority if filter matches.

enumerator `kMCAN_setpriofifo0`
Set priority and store in FIFO 0 if filter matches.

enumerator `kMCAN_setpriofifo1`
Set priority and store in FIFO 1 if filter matches.

enumerator `kMCAN_storeinbuffer`
Store into Rx Buffer or as debug message.

enum `_mcan_std_filter_type`
MCAN Filter Type.
Values:

enumerator `kMCAN_range`
Range filter from SFID1 to SFID2.

enumerator `kMCAN_dual`
Dual ID filter for SFID1 or SFID2.

enumerator `kMCAN_classic`
Classic filter: SFID1 = filter, SFID2 = mask.

enumerator `kMCAN_disableORrange2`
Filter element disabled for standard filter or Range filter, XIDAM mask not applied for extended filter.

typedef enum `_mcan_frame_idformat` `mcan_frame_idformat_t`
MCAN frame format.

typedef enum `_mcan_frame_type` `mcan_frame_type_t`
MCAN frame type.

typedef enum `_mcan_bytes_in_datafield` `mcan_bytes_in_datafield_t`
MCAN frame datafield size.

typedef struct `_mcan_tx_buffer_frame` `mcan_tx_buffer_frame_t`
MCAN Tx Buffer structure.

typedef struct `_mcan_rx_buffer_frame` `mcan_rx_buffer_frame_t`
MCAN Rx FIFO/Buffer structure.

typedef enum `_mcan_fifo_type` `mcan_fifo_type_t`
MCAN Rx FIFO block number.

typedef enum `_mcan_fifo_opmode_config` `mcan_fifo_opmode_config_t`
MCAN FIFO Operation Mode.

typedef enum `_mcan_txmode_config` `mcan_txmode_config_t`
MCAN Tx FIFO/Queue Mode.

```
typedef enum _mcan_remote_frame_config mcan_remote_frame_config_t
    MCAN remote frames treatment.

typedef enum _mcan_nonmasking_frame_config mcan_nonmasking_frame_config_t
    MCAN non-masking frames treatment.

typedef enum _mcan_fec_config mcan_fec_config_t
    MCAN Filter Element Configuration.

typedef struct _mcan_rx_fifo_config mcan_rx_fifo_config_t
    MCAN Rx FIFO configuration.

typedef struct _mcan_rx_buffer_config mcan_rx_buffer_config_t
    MCAN Rx Buffer configuration.

typedef struct _mcan_tx_fifo_config mcan_tx_fifo_config_t
    MCAN Tx Event FIFO configuration.

typedef struct _mcan_tx_buffer_config mcan_tx_buffer_config_t
    MCAN Tx Buffer configuration.

typedef enum _mcan_std_filter_type mcan_filter_type_t
    MCAN Filter Type.

typedef struct _mcan_std_filter_element_config mcan_std_filter_element_config_t
    MCAN Standard Message ID Filter Element.

typedef struct _mcan_ext_filter_element_config mcan_ext_filter_element_config_t
    MCAN Extended Message ID Filter Element.

typedef struct _mcan_frame_filter_config mcan_frame_filter_config_t
    MCAN Rx filter configuration.

typedef struct _mcan_timing_config mcan_timing_config_t
    MCAN protocol timing characteristic configuration structure.

typedef struct _mcan_timing_param mcan_timing_param_t
    MCAN bit timing parameter configuration structure.

typedef struct _mcan_memory_config mcan_memory_config_t
    MCAN Message RAM related configuration structure.

typedef struct _mcan_config mcan_config_t
    MCAN module configuration structure.

typedef struct _mcan_buffer_transfer mcan_buffer_transfer_t
    MCAN Buffer transfer.

typedef struct _mcan_fifo_transfer mcan_fifo_transfer_t
    MCAN Rx FIFO transfer.

typedef struct _mcan_handle mcan_handle_t
    MCAN handle structure definition.

typedef void (*mcan_transfer_callback_t)(CAN_Type *base, mcan_handle_t *handle, status_t
status, uint32_t result, void *userData)
    MCAN transfer callback function.
```

The MCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_MCAN_ErrorStatus`, the result parameter is the Content of MCAN status register which can be used to get the working status(or error status) of MCAN module. If the status equals to other MCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other MCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

MCAN_RETRY_TIMES

struct __mcan_tx_buffer_frame

#include <fsl_mcan.h> MCAN Tx Buffer structure.

Public Members

uint8_t size

classical CAN is 8(bytes), FD is 12/64 such.

struct __mcan_rx_buffer_frame

#include <fsl_mcan.h> MCAN Rx FIFO/Buffer structure.

Public Members

uint8_t size

classical CAN is 8(bytes), FD is 12/64 such.

struct __mcan_rx_fifo_config

#include <fsl_mcan.h> MCAN Rx FIFO configuration.

Public Members

uint32_t address

FIFO start address.

uint32_t elementSize

FIFO element number.

uint32_t watermark

FIFO watermark level.

mcan_fifo_opmode_config_t opmode

FIFO blocking/overwrite mode.

mcan_bytes_in_datafield_t datafieldSize

Data field size per frame, size>8 is for CANFD.

struct __mcan_rx_buffer_config

#include <fsl_mcan.h> MCAN Rx Buffer configuration.

Public Members

uint32_t address

Rx Buffer start address.

mcan_bytes_in_datafield_t datafieldSize

Data field size per frame, size>8 is for CANFD.

struct __mcan_tx_fifo_config

#include <fsl_mcan.h> MCAN Tx Event FIFO configuration.

Public Members

uint32_t address
Event fifo start address.

uint32_t elementSize
FIFO element number.

uint32_t watermark
FIFO watermark level.

struct _mcan_tx_buffer_config
#include <fsl_mcan.h> MCAN Tx Buffer configuration.

Public Members

uint32_t address
Tx Buffers Start Address.

uint32_t dedicatedSize
Number of Dedicated Transmit Buffers.

uint32_t fqSize
Transmit FIFO/Queue Size.

mcan_txmode_config_t mode
Tx FIFO/Queue Mode.

mcan_bytes_in_datafield_t datafieldSize
Data field size per frame, size>8 is for CANFD.

struct _mcan_std_filter_element_config
#include <fsl_mcan.h> MCAN Standard Message ID Filter Element.

Public Members

uint32_t sfid2
Standard Filter ID 2.

uint32_t __pad0__
Reserved.

uint32_t sfid1
Standard Filter ID 1.

uint32_t sfec
Standard Filter Element Configuration.

uint32_t sft
Standard Filter Type.

struct _mcan_ext_filter_element_config
#include <fsl_mcan.h> MCAN Extended Message ID Filter Element.

Public Members

uint32_t efid1
Extended Filter ID 1.

uint32_t efec
Extended Filter Element Configuration.

uint32_t efid2
Extended Filter ID 2.

uint32_t ____pad0____
Reserved.

uint32_t eft
Extended Filter Type.

struct __mcan_frame_filter_config
#include <fsl_mcan.h> MCAN Rx filter configuration.

Public Members

uint32_t address
Filter start address.

uint32_t listSize
Filter list size.

mcan_frame_idformat_t idFormat
Frame format.

mcan_remote_frame_config_t remFrame
Remote frame treatment.

mcan_nonmasking_frame_config_t nmFrame
Non-masking frame treatment.

struct __mcan_timing_config
#include <fsl_mcan.h> MCAN protocol timing characteristic configuration structure.

Public Members

uint16_t preDivider
Nominal Clock Pre-scaler Division Factor.

uint8_t rJumpwidth
Nominal Re-sync Jump Width.

uint8_t seg1
Nominal Time Segment 1.

uint8_t seg2
Nominal Time Segment 2.

uint16_t datapreDivider
Data Clock Pre-scaler Division Factor.

uint8_t datarJumpwidth
Data Re-sync Jump Width.

uint8_t dataseg1
Data Time Segment 1.

uint8_t dataseg2
Data Time Segment 2.

struct `_mcan_timing_param`

#include <fsl_mcan.h> MCAN bit timing parameter configuration structure.

Public Members

uint32_t busLength

Maximum Bus length in meter.

uint32_t propTxRx

Transceiver propagation delay in nanosecond.

uint32_t nominalbaudRate

Baud rate of Arbitration phase in bps.

uint32_t nominalSP

Sample point of Arbitration phase, range in 10 ~ 990, 800 means 80%.

uint32_t databaudRate

Baud rate of Data phase in bps.

uint32_t dataSP

Sample point of Data phase, range in 0 ~ 1000, 800 means 80%.

struct `_mcan_memory_config`

#include <fsl_mcan.h> MCAN Message RAM related configuration structure.

Public Members

uint32_t baseAddr

Message RAM base address, should be 4k alignment.

struct `_mcan_config`

#include <fsl_mcan.h> MCAN module configuration structure.

Public Members

uint32_t baudRateA

Baud rate of Arbitration phase in bps.

uint32_t baudRateD

Baud rate of Data phase in bps.

bool enableCanfdNormal

Enable or Disable CANFD normal.

bool enableCanfdSwitch

Enable or Disable CANFD with baudrate switch.

bool enableLoopBackInt

Enable or Disable Internal Back.

bool enableLoopBackExt

Enable or Disable External Loop Back.

bool enableBusMon

Enable or Disable Bus Monitoring Mode.

mcan_timing_config_t timingConfig

Protocol timing .

struct `_mcan_buffer_transfer`
#include <fsl_mcan.h> MCAN Buffer transfer.

Public Members

`mcan_tx_buffer_frame_t *frame`
The buffer of CAN Message to be transfer.

`uint8_t bufferIdx`
The index of Message buffer used to transfer Message.

struct `_mcan_fifo_transfer`
#include <fsl_mcan.h> MCAN Rx FIFO transfer.

Public Members

`mcan_rx_buffer_frame_t *frame`
The buffer of CAN Message to be received from Rx FIFO.

struct `_mcan_handle`
#include <fsl_mcan.h> MCAN handle structure.

Public Members

`mcan_transfer_callback_t` callback
Callback function.

`void *userData`
MCAN callback function parameter.

`mcan_tx_buffer_frame_t *volatile bufferFrameBuf[64]`
The buffer for received data from Buffers.

`mcan_rx_buffer_frame_t *volatile rxFifoFrameBuf`
The buffer for received data from Rx FIFO.

`volatile uint8_t bufferState[64]`
Message Buffer transfer state.

`volatile uint8_t rxFifoState`
Rx FIFO transfer state.

struct `__unnamed63__`

Public Members

`uint32_t id`
CAN Frame Identifier.

`uint32_t rtr`
CAN Frame Type(DATA or REMOTE).

`uint32_t xtd`
CAN Frame Type(STD or EXT).

`uint32_t esi`
CAN Frame Error State Indicator.

struct `__unnamed65__`

Public Members

uint32_t dlc
Data Length Code 9 10 11 12 13 14 15 Number of data bytes 12 16 20 24 32 48 64

uint32_t brs
Bit Rate Switch.

uint32_t fdf
CAN FD format.

uint32_t __pad1__
Reserved.

uint32_t efc
Event FIFO control.

uint32_t mm
Message Marker.

struct __unnamed67__

Public Members

uint32_t id
CAN Frame Identifier.

uint32_t rtr
CAN Frame Type(DATA or REMOTE).

uint32_t xtd
CAN Frame Type(STD or EXT).

uint32_t esi
CAN Frame Error State Indicator.

struct __unnamed69__

Public Members

uint32_t rxts
Rx Timestamp.

uint32_t dlc
Data Length Code 9 10 11 12 13 14 15 Number of data bytes 12 16 20 24 32 48 64

uint32_t brs
Bit Rate Switch.

uint32_t fdf
CAN FD format.

uint32_t __pad0__
Reserved.

uint32_t fidx
Filter Index.

uint32_t anmf
Accepted Non-matching Frame.

2.61 Memory_interface

Memory group definition.

Values:

enumerator kMemoryGroup_Internal
Memory belongs internal 4G memory region.

enumerator kMemoryGroup_External
Memory belongs external memory region.

Memory device ID definition.

Values:

enumerator kMemoryInternal
Internal memory (include all on chip memory)

enumerator kMemoryQuadSpi0
Qsuad SPI memory 0

enumerator kMemoryIFR0
Nonvolatile information register 0. Only used by SB loader.

enumerator kMemoryFFR
LPCc040hd flash FFR region.

enumerator kMemorySemcNor
SEMC Nor memory

enumerator kMemoryFlexSpiNor
Flex SPI Nor memory

enumerator kMemorySpifiNor
SPIFI Nor memory

enumerator kMemoryFlashExecuteOnly
Execute-only region on internal Flash

enumerator kMemorySemcNand
SEMC NAND memory

enumerator kMemorySpiNand
SPI NAND memory

enumerator kMemorySpiNorEeprom
SPI NOR/EEPROM memory

enumerator kMemoryI2cNorEeprom
I2C NOR/EEPROM memory

enumerator kMemorySDCard
eSD, SD, SDHC, SDXC memory Card

enumerator kMemoryMMCCard
MMC, eMMC memory Card

Bootloader status group numbers.

Values:

enumerator kStatusGroup_Bootloader

Bootloader status group number (100).

enumerator kStatusGroup_MemoryInterface

Memory interface status group number (102).

Memory interface status codes.

Values:

enumerator kStatusMemoryRangeInvalid

Status: Memory Range Invalid

enumerator kStatusMemoryReadFailed

Status: Memory Read Failed

enumerator kStatusMemoryWriteFailed

Status: Memory Write Failed

enumerator kStatusMemoryCumulativeWrite

Status: Memory Cumulative Write

enumerator kStatusMemoryAppOverlapWithExecuteOnlyRegion

Status: Memory AppOverlapWithExecuteOnlyRegion

enumerator kStatusMemoryNotConfigured

Status: Memory Not Configured

enumerator kStatusMemoryAlignmentError

Status: Memory Alignment Error

enumerator kStatusMemoryVerifyFailed

Status: Memory Verify Failed

enumerator kStatusMemoryWriteProtected

Status: Memory Write Protected

enumerator kStatusMemoryAddressError

Status: Memory Address Error

enumerator kStatusMemoryBlankCheckFailed

Status: Memory Blank Check Failed

enumerator kStatusMemoryBlankPageReadDisallowed

Status: Memory Blank Page Read Disallowed

enumerator kStatusMemoryProtectedPageReadDisallowed

Status: Memory Protected Page Read Disallowed

enumerator kStatusMemoryFfrSpecRegionWriteBroken

Status: Memory Ffr Spec Region Write Broken

enumerator kStatusMemoryUnsupportedCommand

Status: Memory Unsupported Command

Bootloader status codes.

Values:

enumerator kStatus_UnknownCommand

Status: Unknown Command

enumerator kStatus_SecurityViolation

Status: Security Violation

enumerator kStatus_AbortDataPhase

Status: Abort Data Phase

enumerator kStatus_Ping

Status: Ping

enumerator kStatus_NoResponse

Status: No Response

enumerator kStatus_NoResponseExpected

Status: No Response Expected

enumerator kStatus_CommandUnsupported

Status: Command Unsupported

typedef union *StandardVersion* standard_version_t

Structure of version property.

typedef struct *kb_api_parameter_struct* kp_api_init_param_t

API initialization data structure.

standard_version_t API_Version(void)

status_t API_Init(*api_core_context_t* *coreCtx, const *kp_api_init_param_t* *param)

Initialize the IAP API runtime environment.

status_t API_Deinit(*api_core_context_t* *coreCtx)

Deinitialize the IAP API runtime environment.

status_t MEM_Init(*api_core_context_t* *coreCtx)

Initialize memory interface.

Return values

- kStatus_Fail –
- kStatus_Success –

status_t MEM_Config(*api_core_context_t* *coreCtx, uint32_t *config, uint32_t memoryId)

Configure memory interface.

Parameters

- coreCtx – A pointer to the storage for the driver runtime state.
- config – memory configure struct.
- memoryId – Indicates the index of the memory type. Please refer to “Memory group definition”

Return values

- kStatus_Success –
- kStatus_CommandUnsupported –
- kStatus_InvalidArgument –
- kStatus_FLASH_ModifyProtectedAreaDisallowed –
- kStatusMemoryRangeInvalid –
- kStatus_Fail –
- kStatus_OutOfRange –

status_t MEM_Write(*api_core_context_t* *coreCtx, uint32_t start, uint32_t lengthInBytes, const uint8_t *buf, uint32_t memoryId)

Write memory.

Parameters

- coreCtx – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. For internal flash the address need to be 512bytes-aligned.
- lengthInBytes – Number of bytes to be programmed.
- buf – A pointer to the source buffer of data that is to be programmed into the flash.
- memoryId – Indicates the index of the memory type. Please refer to “Memory group definition”

Return values

- kStatus__Success –
- kStatus__Fail –
- kStatusMemoryRangeInvalid –
- kStatus__CommandUnsupported –
- kStatus__FLASH_AlignmentError –
- kStatusMemoryCumulativeWrite –
- kStatus__FLASH_InvalidArgument –
- kStatus__FLASH_AddressError –
- kStatus__FLASH_ModifyProtectedAreaDisallowed –
- kStatus__FLASH_CommandFailure –
- kStatus__FLASH_CommandNotSupported –
- kStatus__FLASH_EccError –
- kStatus__FLASH_RegulationLoss –
- kStatus__FLASH_Success –
- kStatus__FLASH_ReadHidingAreaDisallowed –
- kStatus__FLASH_CompareError –
- kStatusMemoryNotConfigured –
- kStatusMemoryVerifyFailed –

status_t MEM_Fill(*api_core_context_t* *coreCtx, uint32_t start, uint32_t lengthInBytes, uint32_t pattern, uint32_t memoryId)

Fill memory with a word pattern.

Parameters

- coreCtx – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. For internal flash the address need to be 512bytes-aligned.
- lengthInBytes – Number of bytes to be programmed.
- pattern – The data to be written into the specified memory area.

- `memoryId` – Indicates the index of the memory type. Please refer to “Memory group definition”

Return values

- `kStatus_CommandUnsupported` –
- `kStatus_Success` –
- `kStatus_FLASH_AlignmentError` –
- `kStatusMemoryCumulativeWrite` –
- `kStatus_Fail` –
- `kStatus_FLASH_InvalidArgument` –
- `kStatus_FLASH_AddressError` –
- `kStatus_FLASH_Success` –
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` –
- `kStatus_FLASH_CommandFailure` –
- `kStatus_FLASH_CommandNotSupported` –
- `kStatus_FLASH_EccError` –
- `kStatus_FLASH_RegulationLoss` –
- `kStatus_FLASH_ReadHidingAreaDisallowed` –

`status_t` MEM_Flush(*api_core_context_t* *coreCtx)

Flush memory.

Return values

- `kStatus_Success` –
- `kStatus_Fail` –
- `kStatusMemoryCumulativeWrite` –
- `kStatus_FLASH_InvalidArgument` –
- `kStatus_FLASH_AlignmentError` –
- `kStatus_FLASH_Success` –
- `kStatus_FLASH_AddressError` –
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` –
- `kStatus_FLASH_CommandFailure` –
- `kStatus_FLASH_CommandNotSupported` –
- `kStatus_FLASH_EccError` –
- `kStatus_FLASH_RegulationLoss` –
- `kStatus_FLASH_ReadHidingAreaDisallowed` –
- `kStatusMemoryVerifyFailed` –

`status_t` MEM_Erase(*api_core_context_t* *coreCtx, `uint32_t` start, `uint32_t` lengthInBytes, `uint32_t` memoryId)

Erase memory.

Parameters

- `coreCtx` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be erased.

- `lengthInBytes` – Number of bytes to be read.
- `memoryId` – Indicates the index of the memory type. Please refer to “Memory group definition”

Return values

- `kStatus_Success` –
- `kStatusMemoryRangeInvalid` –
- `kStatusMemoryAddressError` –
- `kStatus_FLASH_InvalidArgument` –
- `kStatus_FLASH_AlignmentError` –
- `kStatus_FLASH_Success` –
- `kStatus_FLASH_AddressError` –
- `kStatus_FLASH_EraseKeyError` –
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` –
- `kStatus_Fail` –
- `kStatus_FLASH_CommandFailure` –
- `kStatus_FLASH_CommandNotSupported` –
- `kStatus_FLASH_EccError` –
- `kStatus_FLASH_RegulationLoss` –
- `kStatusMemoryNotConfigured` –
- `kStatusMemoryVerifyFailed` –

status_t MEM_EraseAll(*api_core_context_t* *coreCtx, uint32_t memoryId)

Erase entire memory based on memoryId.

Parameters

- `coreCtx` – A pointer to the storage for the driver runtime state.
- `memoryId` – Indicates the index of the memory type. Please refer to “Memory group definition”

Return values

- `kStatus_Success` –
- `kStatus_Fail` –
- `kStatus_CommandUnsupported` –
- `kStatus_FLASH_InvalidArgument` –
- `kStatus_FLASH_AlignmentError` –
- `kStatus_FLASH_Success` –
- `kStatus_FLASH_AddressError` –
- `kStatus_FLASH_EraseKeyError` –
- `kStatus_FLASH_CommandFailure` –
- `kStatus_FLASH_CommandNotSupported` –
- `kStatus_FLASH_EccError` –
- `kStatus_FLASH_RegulationLoss` –
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` –

- kStatusMemoryVerifyFailed –
- kStatusMemoryNotConfigured –
- kStatus_InvalidArgument –

FSL_ROMAPI_MEM_INTERFACE_DRIVER_VERSION

ROMAPI_MEM_INTERFACE driver version 2.0.0.

DEVICE_ID_MASK

Bit mask for device ID.

DEVICE_ID_SHIFT

Bit position of device ID.

GROUP_ID_MASK

Bit mask for group ID.

GROUP_ID_SHIFT

Bit position of group ID.

MAKE_MEMORYID(group, device)

Construct a memory ID from a given group ID and device ID.

GROUPID(memoryId)

Get group ID from a given memory ID.

DEVICEID(memoryId)

Get device ID from a given memory ID.

status_t (*init)(void)

status_t (*read)(uint32_t address, uint32_t length, uint8_t *buffer, uint32_t memoryId)

status_t (*write)(uint32_t address, uint32_t length, const uint8_t *buffer, uint32_t memoryId)

status_t (*fill)(uint32_t address, uint32_t length, uint32_t pattern)

status_t (*flush)(void)

status_t (*finalize)(void)

status_t (*erase)(uint32_t address, uint32_t length, uint32_t memoryId)

status_t (*init)(void)

status_t (*read)(uint32_t address, uint32_t length, uint8_t *buffer)

status_t (*write)(uint32_t address, uint32_t length, const uint8_t *buffer)

status_t (*fill)(uint32_t address, uint32_t length, uint32_t pattern)

status_t (*flush)(void)

status_t (*erase)(uint32_t address, uint32_t length)

status_t (*config)(uint32_t *buffer)

status_t (*erase_all)(void)

uint32_t startAddress

uint32_t endAddress

uint32_t memoryProperty


```
uint32_t memoryId
const memory_region_interface_t *memoryInterface
uint8_t bugfix
    bugfix version [7:0]
uint8_t minor
    minor version [15:8]
uint8_t major
    major version [23:16]
char name
    name [31:24]
struct StandardVersion
uint32_t version
    combined version numbers
uint32_t allocStart
uint32_t allocSize
struct memory__interface__t
    #include <fsl_mem_interface.h> Interface to memory operations.
    This is the main abstract interface to all memory operations.
struct memory__region__interface__t
    #include <fsl_mem_interface.h> Interface to memory operations for one region of memory.
struct memory__map__entry__t
    #include <fsl_mem_interface.h> Structure of a memory map entry.
union StandardVersion
    #include <fsl_mem_interface.h> Structure of version property.
struct kb_api__parameter__struct
    #include <fsl_mem_interface.h> API initialization data structure.
struct ____unnamed33____
```

Public Members

```
uint8_t bugfix
    bugfix version [7:0]
uint8_t minor
    minor version [15:8]
uint8_t major
    major version [23:16]
char name
    name [31:24]
```

2.62 MRT: Multi-Rate Timer

`void MRT_Init(MRT_Type *base, const mrt_config_t *config)`

Ungates the MRT clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the MRT driver.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `config` – Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MODCFG register, param config is useless.

`void MRT_Deinit(MRT_Type *base)`

Gate the MRT clock.

Parameters

- `base` – Multi-Rate timer peripheral base address

`static inline void MRT_GetDefaultConfig(mrt_config_t *config)`

Fill in the MRT config struct with the default settings.

The default values are:

```
config->enableMultiTask = false;
```

Parameters

- `config` – Pointer to user's MRT config structure.

`static inline void MRT_SetupChannelMode(MRT_Type *base, mrt_chnl_t channel, const mrt_timer_mode_t mode)`

Sets up an MRT channel mode.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Channel that is being configured.
- `mode` – Timer mode to use for the channel.

`static inline void MRT_EnableInterrupts(MRT_Type *base, mrt_chnl_t channel, uint32_t mask)`

Enables the MRT interrupt.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `mrt_interrupt_enable_t`

`static inline void MRT_DisableInterrupts(MRT_Type *base, mrt_chnl_t channel, uint32_t mask)`

Disables the selected MRT interrupt.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number

- `mask` – The interrupts to disable. This is a logical OR of members of the enumeration `mrt_interrupt_enable_t`

static inline uint32_t MRT_GetEnabledInterrupts(MRT_Type *base, *mrt_chnl_t* channel)

Gets the enabled MRT interrupts.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `mrt_interrupt_enable_t`

static inline uint32_t MRT_GetStatusFlags(MRT_Type *base, *mrt_chnl_t* channel)

Gets the MRT status flags.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration `mrt_status_flags_t`

static inline void MRT_ClearStatusFlags(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Clears the MRT status flags.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `mrt_status_flags_t`

void MRT_UpdateTimerPeriod(MRT_Type *base, *mrt_chnl_t* channel, uint32_t count, bool immediateLoad)

Used to update the timer period in units of count.

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

Note: User can call the utility macros provided in `fsl_common.h` to convert to ticks

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number
- `count` – Timer period in units of ticks
- `immediateLoad` – `true`: Load the new value immediately into the TIMER register; `false`: Load the new value at the end of current timer interval

static inline uint32_t MRT_GetCurrentTimerCount(MRT_Type *base, *mrt_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number

Returns

Current timer counting value in ticks

`static inline void MRT_StartTimer(MRT_Type *base, mrt_chnl_t channel, uint32_t count)`

Starts the timer counting.

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

Note: User can call the utility macros provided in `fsl_common.h` to convert to ticks

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number.
- `count` – Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

`static inline void MRT_StopTimer(MRT_Type *base, mrt_chnl_t channel)`

Stops the timer counting.

This function stops the timer from counting.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number.

`static inline uint32_t MRT_GetIdleChannel(MRT_Type *base)`

Find the available channel.

This function returns the lowest available channel number.

Parameters

- `base` – Multi-Rate timer peripheral base address

`static inline void MRT_ReleaseChannel(MRT_Type *base, mrt_chnl_t channel)`

Release the channel when the timer is using the multi-task mode.

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling `MRT_GetIdleChannel()` for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

Parameters

- `base` – Multi-Rate timer peripheral base address
- `channel` – Timer channel number.

`FSL_MRT_DRIVER_VERSION`

enum `_mrt_chnl`

List of MRT channels.

Values:

enumerator `kMRT_Channel_0`

MRT channel number 0

enumerator `kMRT_Channel_1`

MRT channel number 1

enumerator `kMRT_Channel_2`

MRT channel number 2

enumerator `kMRT_Channel_3`

MRT channel number 3

enum `_mrt_timer_mode`

List of MRT timer modes.

Values:

enumerator `kMRT_RepeatMode`

Repeat Interrupt mode

enumerator `kMRT_OneShotMode`

One-shot Interrupt mode

enumerator `kMRT_OneShotStallMode`

One-shot stall mode

enum `_mrt_interrupt_enable`

List of MRT interrupts.

Values:

enumerator `kMRT_TimerInterruptEnable`

Timer interrupt enable

enum `_mrt_status_flags`

List of MRT status flags.

Values:

enumerator `kMRT_TimerInterruptFlag`

Timer interrupt flag

enumerator `kMRT_TimerRunFlag`

Indicates state of the timer

typedef enum `_mrt_chnl` `mrt_chnl_t`

List of MRT channels.

typedef enum `_mrt_timer_mode` `mrt_timer_mode_t`

List of MRT timer modes.

typedef enum `_mrt_interrupt_enable` `mrt_interrupt_enable_t`

List of MRT interrupts.

typedef enum `_mrt_status_flags` `mrt_status_flags_t`

List of MRT status flags.

```
typedef struct _mrt_config mrt_config_t
```

MRT configuration structure.

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the `MRT_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
struct __mrt_config
```

#include <fsl_mrt.h> MRT configuration structure.

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the `MRT_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 enableMultiTask`

true: Timers run in multi-task mode; false: Timers run in hardware status mode

2.63 Nboot

nboot status codes.

Values:

enumerator `kStatus_NBOOT_Success`

Operation completed successfully.

enumerator `kStatus_NBOOT_Fail`

Operation failed.

enumerator `kStatus_NBOOT_InvalidArgument`

Invalid argument passed to the function.

enumerator `kStatus_NBOOT_RequestTimeout`

Operation timed out.

enumerator `kStatus_NBOOT_KeyNotLoaded`

The requested key is not loaded.

enumerator `kStatus_NBOOT_AuthFail`

Authentication failed.

enumerator `kStatus_NBOOT_OperationNotAvaialable`

Operation not available on this HW.

enumerator `kStatus_NBOOT_KeyNotAvailable`

Key is not avaialble.

enumerator `kStatus_NBOOT_IvCounterOverflow`

Overflow of IV counter (PRINCE/IPED).

enumerator `kStatus_NBOOT_SelftestFail`

FIPS self-test failure.

enumerator kStatus_NBOOT_InvalidDataFormat

Invalid data format for example antipole

enumerator kStatus_NBOOT_IskCertUserDataTooBig

Size of User data in ISK certificate is greater than 96 bytes

enumerator kStatus_NBOOT_IskCertSignatureOffsetTooSmall

Signature offset in ISK certificate is smaller than expected

enumerator kStatus_NBOOT_MemcpyFail

Unexpected error detected during nboot_memcpy()

enum __nboot_bool

Boolean type for the NBOOT functions.

This type defines boolean values used by NBOOT functions that are not easily disturbed by Fault Attacks

Values:

enumerator kNBOOT_TRUE

Value for TRUE.

enumerator kNBOOT_TRUE256

Value for TRUE when P256 was used to sign the image.

enumerator kNBOOT_TRUE384

Value for TRUE when P384 was used to sign the image.

enumerator kNBOOT_FALSE

Value for FALSE.

enumerator kNBOOT_OperationAllowed

enumerator kNBOOT_OperationDisallowed

typedef int romapi_status_t

typedef uint32_t nboot_root_key_usage_t

typedef uint32_t nboot_root_key_revocation_t

typedef uint32_t nboot_root_key_type_and_length_t

typedef uint32_t nboot_soc_lifecycle_t

typedef uint32_t nboot_status_t

Type for nboot status codes.

typedef uint64_t nboot_status_protected_t

Type for nboot protected status codes.

typedef struct *nboot_secure_counter* nboot_secure_counter_t

Data structure holding secure counter value used by nboot library.

typedef struct *nboot_context* nboot_context_t

NBOOT context type.

This type defines the NBOOT context

typedef struct *nboot_rot_auth_params* nboot_rot_auth_params_t

NBOOT type for the root of trust parameters.

This type defines the NBOOT root of trust parameters

`typedef struct _nboot_sb3_load_manifest_parms` `nboot_sb3_load_manifest_parms_t`
manifest loading parameters

This type defines the NBOOT SB3.1 manifest loading parameters

`typedef struct _nboot_img_auth_ecdsa_parms` `nboot_img_auth_ecdsa_parms_t`
Data structure holding input arguments to POR secure boot (authentication) algorithm. Shall be read from SoC trusted NVM or SoC fuses.

`typedef struct _nboot_cmac_authenticate_parms` `nboot_img_authenticate_cmac_parms_t`
Data structure holding input arguments for CMAC authentication.

`typedef enum _nboot_bool` `nboot_bool_t`
Boolean type for the NBOOT functions.

This type defines boolean values used by NBOOT functions that are not easily disturbed by Fault Attacks

`status_t` `NBOOT_GenerateRandom(uint8_t *output, size_t outputByteLen)`
This API function is used to generate random number with specified length.

Parameters

- `output` – Pointer to random number buffer
- `outputByteLen` – length of generated random number in bytes. Length has to be in range $<1, 2^{16}>$

Return values

- `kStatus_NBOOT_InvalidArgument` – Invalid input parameters (Input pointers points to NULL or length is invalid)
- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

`nboot_status_t` `NBOOT_ContextInit(nboot_context_t *context)`

The function is used for initializing of the nboot context data structure. It should be called prior to any other calls of nboot API.

Parameters

- `context` – Pointer to `nboot_context_t` structure.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

`nboot_status_t` `NBOOT_ContextDeinit(nboot_context_t *context)`

The function is used to deinitialize nboot context data structure. Its contents are overwritten with random data so that any sensitive data does not remain in memory.

Parameters

- `context` – Pointer to `nboot_context_t` structure.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

`nboot_status_protected_t` `NBOOT_Sb3LoadManifest(nboot_context_t *context, uint32_t *manifest, nboot_sb3_load_manifest_parms_t *params)`

Verify NBOOT SB3.1 manifest (header message)

This function verifies the NBOOT SB3.1 manifest (header message), initializes the context and loads keys into the CSS key store so that they can be used by `nboot_sb3_load_block` function. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function. Please note that this API is intended to be used only by users who needs to split FW update process (loading of SB3.1 file) to partial steps to customize whole operation. For regular SB3.1 processing, please use API described in chapter SBloader APIs.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `manifest` – Pointer to the input manifest buffer
- `params` – additional input parameters. Please refer to `nboot_sb3_load_manifest_parms_t` definition for details.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

nboot_status_protected_t NBOOT_Sb3LoadBlock(*nboot_context_t* *context, uint32_t *block)

Verify NBOOT SB3.1 block.

This function verifies and decrypts an NBOOT SB3.1 block. Decryption is performed in-place. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function. Please note that this API is intended to be used only by users who needs to split FW update process (loading of SB3.1 file) to partial steps to customize whole operation. For regular SB3.1 processing, please use API described in chapter SBloader APIs.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `block` – Pointer to the input SB3.1 data block

Return values

- `kStatus_NBOOT_Success` – successfully finished
- `kStatus_NBOOT_Fail` – occurred during operation

nboot_status_protected_t NBOOT_ImgAuthenticateEcdsa(*nboot_context_t* *context, uint8_t imageStartAddress[], *nboot_bool_t* *isSignatureVerified, *nboot_img_auth_ecdsa_parms_t* *parms)

This function authenticates image with asymmetric cryptography. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `imageStartAddress` – Pointer to start of the image in memory.
- `isSignatureVerified` – Pointer to memory holding function call result. After the function returns, the value will be set to `kNBOOT_TRUE` when the image is authentic. Any other value means the authentication does not pass.
- `parms` – Pointer to a data structure in trusted memory, holding input parameters for the algorithm. The data structure shall be correctly filled before the function call.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Returned in all other cases. Doesn't always mean invalid image, it could also mean transient error caused by short time environmental conditions.

`nboot_status_protected_t` `NBOOT_ImgAuthenticateCmac`(`nboot_context_t` *context, `uint8_t` imageStartAddress[], `nboot_bool_t` *isSignatureVerified, `nboot_img_authenticate_cmac_parms_t` *parms)

This function calculates the CMAC over the given image and compares it to the expected value. To be more resistant against SPA, it is recommended that imageStartAddress is word aligned. The NBOOT context has to be initialized by the `nboot_context_init()` before calling this function.

Parameters

- context – Pointer to `nboot_context_t` structure.
- imageStartAddress – Pointer to start of the image in memory.
- isSignatureVerified – Pointer to memory holding function call result. After the function returns, the value will be set to
- parms – Pointer to a data structure in trusted memory, holding the reference MAC. The data structure shall be correctly filled before the function call.

Return values

- `kStatus_NBOOT_Success` –
- `kStatus_NBOOT_Fail` –

`FSL_ROMAPI_NBOOT_DRIVER_VERSION`

ROMAPI_NBOOT driver version 2.0.0.

`NXPCLHASH_WA_SIZE_MAX`

Define the max workarea size required for this component.

`NBOOT_ROOT_CERT_COUNT`

`NXPCLCSS_HASH_RTF_OUTPUT_SIZE_HAL`

Size of RTF appendix to hash output buffer, in bytes.

`NBOOT_KEYINFO_WORDLEN`

`NBOOT_CONTEXT_BYTELEN`

`NBOOT_CONTEXT_WORDLEN`

`kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey`

NBOOT type for the root key usage.

This type defines the NBOOT root key usage; any other value means the root key is not valid (treat as if revoked).

`kNBOOT_RootKeyUsage_DebugCA`

`kNBOOT_RootKeyUsage_ImageCA_FwCA`

`kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA`

`kNBOOT_RootKeyUsage_ImageKey_FwKey`

kNBOOT_RootKeyUsage_ImageKey

kNBOOT_RootKeyUsage_FwKey

kNBOOT_RootKeyUsage_Unused

kNBOOT_RootKey_Enabled

NBOOT type for the root key revocation.

This type defines the NBOOT root key revocation; any other value means the root key is revoked.

kNBOOT_RootKey_Revoked

kNBOOT_RootKey_Ecdsa_P256

NBOOT type specifying the elliptic curve to be used.

This type defines the elliptic curve type and length

kNBOOT_RootKey_Ecdsa_P384

nboot_lc_nxpBlank

Enumeration for SoC Lifecycle.

nboot_lc_nxpFab

nboot_lc_nxpDev

nboot_lc_nxpProvisioned

nboot_lc_oemOpen

nboot_lc_oemSecureWorld

nboot_lc_oemClosed

nboot_lc_oemLocked

nboot_lc_oemFieldReturn

nboot_lc_nxpFieldReturn

nboot_lc_shredded

uint32_t sc

uint32_t scAp

uint32_t totalBlocks

holds number of SB3 blocks. Initialized by nboot_sb3_load_header().

uint32_t processData

flag, initialized by nboot_sb3_load_header(). SB3 related flag set by NBOOT in case the nboot_sb3_load_block() provides plain data to output buffer (for processing by ROM SB3 loader)

uint32_t timeout

timeout value for css operation. In case it is 0, infinite wait is performed

uint32_t keyinfo[(23U)]

data for NBOOT key management.

uint32_t context[((192U + (128U + 64U)) / sizeof(uint32_t))]

work area for NBOOT lib.

uint32_t uuid[4]

holds UUID value from NMPA

uint32_t prngReadyFlag

flag, used by `nboot_rng_generate_lq_random()` to determine whether CSS is ready to generate rnd number

uint32_t multipartMacBuffer[1024 / sizeof(uint32_t)]

uint32_t oemShareValidFlag

flag, used during TP to determine whether valid oemShare was set by `nboot_tp_isp_gen_oem_master_share()`

uint32_t oemShare[4]

buffer to store OEM_SHARE computed by `npxCLTrustProv_nboot_isp_gen_oem_master_share()`

nboot_secure_counter_t secureCounter

Secure counter used by nboot

uint32_t rtf[((size_t)32U) / sizeof(uint32_t)]

uint32_t imageHash[48 / sizeof(uint32_t)]

uint32_t authStatus

nboot_root_key_revocation_t soc_rootKeyRevocation[(4U)]

Provided by caller based on NVM information in CFPA: ROTKH_REVOKE

uint32_t soc_imageKeyRevocation

Provided by caller based on NVM information in CFPA: IMAGE_KEY_REVOKE

uint32_t soc_rkh[12]

Provided by caller based on NVM information in CMPA: ROTKH (hash of hashes)
In case of `kNBOOT_RootKey_Ecdsa_P384`, `sock_rkh[0..11]` are used In case of `kNBOOT_RootKey_Ecdsa_P256`, `sock_rkh[0..7]` are used

uint32_t soc_numberOfRootKeys

unsigned int, between minimum = 1 and maximum = 4;

nboot_root_key_usage_t soc_rootKeyUsage[(4U)]

CMPA

nboot_root_key_type_and_length_t soc_rootKeyTypeAndLength

static selection between ECDSA P-256 or ECDSA P-384 based root keys

nboot_soc_lifecycle_t soc_lifecycle

nboot_rot_auth_parms_t soc_RoTNVM

trusted information originated from CFPA and NMPA

uint32_t soc_trustedFirmwareVersion

Provided by caller based on NVM information in CFPA: Secure_FW_Version

uint8_t pckBlob[48]

nboot_rot_auth_parms_t soc_RoTNVM

trusted information originated from CFPA and NMPA

uint32_t soc_trustedFirmwareVersion

Provided by caller based on NVM information in CFPA: Secure_FW_Version

uint32_t expectedMAC[4]

expected MAC result

struct __nboot_secure_counter

#include <fsl_nboot.h> Data structure holding secure counter value used by nboot library.

struct __nboot_context

#include <fsl_nboot.h> NBOOT context type.

This type defines the NBOOT context

struct __nboot_rot_auth_parms

#include <fsl_nboot.h> NBOOT type for the root of trust parameters.

This type defines the NBOOT root of trust parameters

struct __nboot_sb3_load_manifest_parms

#include <fsl_nboot.h> manifest loading parameters

This type defines the NBOOT SB3.1 manifest loading parameters

struct __nboot_img_auth_ecdsa_parms

#include <fsl_nboot.h> Data structure holding input arguments to POR secure boot (authentication) algorithm. Shall be read from SoC trusted NVM or SoC fuses.

struct __nboot_cmac_authenticate_parms

#include <fsl_nboot.h> Data structure holding input arguments for CMAC authentication.

2.64 Nboot_hal

enum __nboot_hash_algo_t

Algorithm used for nboot HASH operation.

Values:

enumerator kHASH_Sha1

SHA_1

enumerator kHASH_Sha256

SHA_256

enumerator kHASH_Sha512

SHA_512

enumerator kHASH_Aes

AES

enumerator kHASH_AesIcb

AES_ICB

typedef uint32_t nboot_timestamp_t[2]

NBOOT type for a timestamp.

This type defines the NBOOT timestamp

typedef struct __nboot_sb3_header nboot_sb3_header_t

NBOOT SB3.1 header type.

This type defines the header used in the SB3.1 manifest

typedef struct __nboot_certificate_header_block nboot_certificate_header_block_t

NBOOT type for the header of the certificate block.

This type defines the NBOOT header of the certificate block, it is part of the nboot_certificate_block_t

```
typedef uint8_t nboot_ctrk_hash_t[(48u)]
```

```
typedef struct nboot_ctrk_hash_table nboot_ctrk_hash_table_t
```

NBOOT type for the hash table.

This type defines the NBOOT hash table

```
typedef uint8_t nboot_ecc_coordinate_t[(48u)]
```

NBOOT type for an ECC coordinate.

This type defines the NBOOT ECC coordinate type ECC point coordinate, up to 384-bits. big endian.

```
typedef struct nboot_root_certificate_block nboot_root_certificate_block_t
```

NBOOT type for the root certificate block.

This type defines the NBOOT root certificate block, it is part of the `nboot_certificate_block_t`

```
typedef struct nboot_certificate_block nboot_certificate_block_t
```

NBOOT type for the certificate block.

This type defines the constant length part of an NBOOT certificate block

```
typedef enum nboot_hash_algo_t nboot_hash_algo_t
```

Algorithm used for nboot HASH operation.

```
NBOOT_UUID_SIZE_IN_WORD
```

The size of the UUID.

```
NBOOT_UUID_SIZE_IN_BYTE
```

```
NBOOT_PUF_AC_SIZE_IN_BYTE
```

The size of the PUF activation code.

```
NBOOT_PUF_KC_SIZE_IN_BYTE
```

The size of the PUF key code.

```
NBOOT_KEY_STORE_SIZE_IN_BYTE
```

The size of the key store.

```
NBOOT_ROOT_ROT_KH_SIZE_IN_WORD
```

The size of the root of trust key table hash.

```
NBOOT_ROOT_ROT_KH_SIZE_IN_BYTE
```

```
NBOOT_KEY_BLOB_SIZE_IN_BYTE_256
```

The size of the blob with Key Blob.

```
NBOOT_KEY_BLOB_SIZE_IN_BYTE_384
```

```
NBOOT_KEY_BLOB_SIZE_IN_BYTE_MAX
```

```
NBOOT_DBG_AUTH_DBG_STATE_MASK
```

The mask of the value of the debug state .

```
NBOOT_DBG_AUTH_DBG_STATE_SHIFT
```

The shift inverted value of the debug state.

```
NBOOT_DBG_AUTH_DBG_STATE_ALL_DISABLED
```

The value with all debug feature disabled.

```
NBOOT_ROOT_OF_TRUST_HASH_SIZE_IN_BYTES
```

```
NBOOT_EC_COORDINATE_384_SIZE_IN_BYTES
```

NBOOT_EC_COORDINATE_MAX_SIZE

NBOOT_SB3_CHUNK_SIZE_IN_BYTES

NBOOT_SB3_BLOCK_HASH256_SIZE_IN_BYTES

NBOOT_SB3_BLOCK_HASH384_SIZE_IN_BYTES

NBOOT_SB3_MANIFEST_MAX_SIZE_IN_BYTES

NBOOT_SB3_BLOCK_MAX_SIZE_IN_BYTES

NBOOT_DICE_CSR_SIZE_IN_WORD

The size of the DICE certificate.

NBOOT_DICE_CSR_SIZE_IN_BYTES

NBOOT_DICE_CSR_ADDRESS

The physical address to put the DICE certificate.

NBOOT_IPED_IV_OFFSET

The offset for the PRCINE/IPED erase region return by nboot mem checker.

NBOOT_IMAGE_CMAC_UPDATE_NONE

NBOOT_IMAGE_CMAC_UPDATE_INDEX0

NBOOT_IMAGE_CMAC_UPDATE_INDEX1

NBOOT_IMAGE_CMAC_UPDATE_BOTH

NBOOT_IMAGE_CMAC_UPDATE_MASK

NBOOT_CMPA_CMAC_UPDATE_MASK

NBOOT_CMPA_CMAC_UPDATE_SHIFT

NBOOT_CMPA_UPDATE_CMAC_PFR

NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_SECURE

NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_CLOSE

NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_LOCKED

struct _nboot_sb3_header

#include <fsl_nboot_hal.h> NBOOT SB3.1 header type.

This type defines the header used in the SB3.1 manifest

Public Members

uint32_t magic

offset 0x00: Fixed 4-byte string of 'sbv3' without the trailing NULL

uint32_t formatVersion

offset 0x04: (major = 3, minor = 1); The format version determines the manifest (block0) size.

uint32_t flags

offset 0x08: not defined yet, keep zero for future compatibility

uint32_t blockCount

offset 0x0C: Number of blocks not including the manifest (block0).

uint32_t blockSize

offset 0x10: Size in bytes of data block (repeated blockCount times for SB3 data stream).

nboot_timestamp_t timeStamp

offset 0x14: 64-bit value used as key derivation data.

uint32_t firmwareVersion

offset 0x1c: Version number of the included firmware

uint32_t imageTotalLength

offset 0x20: Total manifest length in bytes, including signatures etc.

uint32_t imageType

offset 0x24: image type and flags

uint32_t certificateBlockOffset

offset 0x28: Offset from start of header block to the certificate block.

uint8_t description[16]

offset 0x32: This field provides description of the file. It is an arbitrary string injected by the signing tool, which helps to identify the file.

struct __nboot_certificate_header_block

#include <fsl_nboot_hal.h> NBOOT type for the header of the certificate block.

This type defines the NBOOT header of the certificate block, it is part of the nboot_certificate_block_t

Public Members

uint32_t magic

magic number.

uint32_t formatMajorMinorVersion

format major minor version

uint32_t certBlockSize

Size of the full certificate block

struct __nboot_ctrk_hash_table

#include <fsl_nboot_hal.h> NBOOT type for the hash table.

This type defines the NBOOT hash table

struct nboot_ecdsa_public_key_t

#include <fsl_nboot_hal.h> NBOOT type for an ECC point.

This type defines the NBOOT ECC point type

Public Members

nboot_ecc_coordinate_t x

x portion of the ECDSA public key, up to 384-bits. big endian.

nboot_ecc_coordinate_t y

y portion of the ECDSA public key, up to 384-bits. big endian.

struct _nboot_root_certificate_block

#include <fsl_nboot_hal.h> NBOOT type for the root certificate block.

This type defines the NBOOT root certificate block, it is part of the nboot_certificate_block_t

Public Members

uint32_t flags

root certificate flags

nboot_ctrk_hash_table_t ctrkHashTable

hash table

nboot_ecdsa_public_key_t rootPublicKey

root public key

struct nboot_ecdsa_signature_t

#include <fsl_nboot_hal.h> NBOOT type for an ECC signature.

This type defines the NBOOT ECC signature type

Public Members

nboot_ecc_coordinate_t r

r portion of the ECDSA signature, up to 384-bits. big endian.

nboot_ecc_coordinate_t s

s portion of the ECDSA signature, up to 384-bits. big endian.

struct nboot_isk_block_t

#include <fsl_nboot_hal.h> NBOOT type for the isk block.

This type defines the constant length part of an NBOOT isk block

Public Members

uint32_t signatureOffset

Offset of signature in ISK block.

uint32_t constraints

Version number of signing certificate.

uint32_t iskFlags

Reserved for definiton of ISK certificate flags.

nboot_ecdsa_public_key_t iskPubKey

Public key of signing certificate. Variable length; only used to determine start address

nboot_ecdsa_public_key_t userData

Space for at lest one addition public key

nboot_ecdsa_signature_t iskSign

ISK signature

struct _nboot_certificate_block

#include <fsl_nboot_hal.h> NBOOT type for the certificate block.

This type defines the constant length part of an NBOOT certificate block

Public Members

nboot_isk_block_t iskBlock

Details of selected root certificate (root certificate which will be used for ISK signing/SB3 header signing)

2.65 OPAMP: Operational Amplifier

`void OPAMP_Init(OPAMP_Type *base, const opamp_config_t *config)`

Initialize OPAMP instance.

Parameters

- *base* – OPAMP peripheral base address.
- *config* – The pointer to *opamp_config_t*.

`void OPAMP_Deinit(OPAMP_Type *base)`

De-initialize OPAMP instance.

Parameters

- *base* – OPAMP peripheral base address.

`void OPAMP_GetDefaultConfig(opamp_config_t *config)`

Get default configuration of OPAMP.

```
config->enable      = false;
config->mode        = kOPAMP_LowNoiseMode;
config->trimOption   = kOPAMP_TrimOptionDefault;
config->intRefVoltage = kOPAMP_IntRefVoltVddaDiv2;
config->enablePosADCSw = false;
config->posRefVoltage = kOPAMP_PosRefVoltVrefh3;
config->posGain      = kOPAMP_PosGainReserved;
config->negGain      = kOPAMP_NegGainBufferMode;
```

Parameters

- *config* – The pointer to *opamp_config_t*.

`static inline void OPAMP_DoPosGainConfig(OPAMP_Type *base, opamp_positive_gain_t option)`

Configure OPAMP positive port gain.

Parameters

- *base* – OPAMP peripheral base address.
- *option* – OPAMP positive port gain.

`static inline void OPAMP_DoNegGainConfig(OPAMP_Type *base, opamp_negative_gain_t option)`

Configure OPAMP negative port gain.

Parameters

- *base* – OPAMP peripheral base address.
- *option* – OPAMP negative port gain.

`FSL_OPAMP_DRIVER_VERSION`

OPAMP driver version.

enum _opamp_mode

The enumeration of OPAMP mode, including low noise mode and high speed mode.

Values:

enumerator kOPAMP_LowNoiseMode

Set opamp mode as low noise mode.

enumerator kOPAMP_HighSpeedMode

Set opamp mode as high speed mode.

enum _opamp_bias_current_trim_option

The enumeration of bias current trim option.

Values:

enumerator kOPAMP_TrimOptionDefault

Default Bias current trim option.

enumerator kOPAMP_TrimOptionIncreaseCurrent

Trim option selected as increase current.

enumerator kOPAMP_TrimOptionDecreaseCurrent

Trim option selected as decrease current.

enumerator kOPAMP_TrimOptionFurtherDecreaseCurrent

Trim option selected as further decrease current.

enum _opamp_internal_ref_voltage

The enumeration of internal reference voltage.

Values:

enumerator kOPAMP_IntRefVoltVddaDiv2

Internal reference voltage selected as Vdda/2.

enumerator kOPAMP_IntRefVoltVdda3V

Internal reference voltage selected as Vdda_3V.

enumerator kOPAMP_IntRefVoltVssa3V

Internal reference voltage selected as Vssa_3V.

enumerator kOPAMP_IntRefVoltNotAllowed

Internal reference voltage not allowed.

enum _opamp_positive_ref_voltage

The enumeration of positive reference voltage—please refer to manual use—.

Values:

enumerator kOPAMP_PosRefVoltVrefh3

Positive part reference voltage select Vrefh3, connected from DAC output.

enumerator kOPAMP_PosRefVoltVrefh0

Positive part reference voltage select Vrefh0, connected from VDDA supply.

enumerator kOPAMP_PosRefVoltVrefh1

Positive part reference voltage select Vrefh1, connected from Voltage reference output.

enumerator kOPAMP_PosRefVoltVrefh4

Positive part reference voltage select 520mv or reserved.

enum `_opamp_positive_gain`

The enumeration of positive programmable gain (please refer to manual use).

Values:

enumerator `kOPAMP_PosGainReserved`

Positive Gain reserved.

enumerator `kOPAMP_PosGainNonInvert1X`

Positive non-inverting gain application 1X.

enumerator `kOPAMP_PosGainNonInvert2X`

Positive non-inverting gain application 2X.

enumerator `kOPAMP_PosGainNonInvert4X`

Positive non-inverting gain application 4X.

enumerator `kOPAMP_PosGainNonInvert8X`

Positive non-inverting gain application 8X.

enumerator `kOPAMP_PosGainNonInvert16X`

Positive non-inverting gain application 16X.

enumerator `kOPAMP_PosGainNonInvert33X`

Positive non-inverting gain application 33X.

enumerator `kOPAMP_PosGainNonInvert64X`

Positive non-inverting gain application 64X.

enum `_opamp_negative_gain`

The enumeration of negative programmable gain.

Values:

enumerator `kOPAMP_NegGainBufferMode`

Negative Buffer Mode.

enumerator `kOPAMP_NegGainInvert1X`

Negative inverting gain application -1X.

enumerator `kOPAMP_NegGainInvert2X`

Negative inverting gain application -2X.

enumerator `kOPAMP_NegGainInvert4X`

Negative inverting gain application -4X.

enumerator `kOPAMP_NegGainInvert8X`

Negative inverting gain application -8X.

enumerator `kOPAMP_NegGainInvert16X`

Negative inverting gain application -16X.

enumerator `kOPAMP_NegGainInvert33X`

Negative inverting gain application -33X.

enumerator `kOPAMP_NegGainInvert64X`

Negative inverting gain application -64X.

typedef enum `_opamp_mode` `opamp_mode_t`

The enumeration of OPAMP mode, including low noise mode and high speed mode.

typedef enum `_opamp_bias_current_trim_option` `opamp_bias_current_trim_option_t`

The enumeration of bias current trim option.

```
typedef enum _opamp_internal_ref_voltage opamp_internal_ref_voltage_t
```

The enumeration of internal reference voltage.

```
typedef enum _opamp_positive_ref_voltage opamp_positive_ref_voltage_t
```

The enumeration of positive reference voltage—please refer to manual use—.

```
typedef enum _opamp_positive_gain opamp_positive_gain_t
```

The enumeration of positive programmable gain (please refer to manual use).

```
typedef enum _opamp_negative_gain opamp_negative_gain_t
```

The enumeration of negative programmable gain.

```
typedef struct _opamp_config opamp_config_t
```

OPAMP configuraion, including mode, internal reference voltage, positive gain, negative gain and so on.

```
struct _opamp_config
```

#include <fsl_opamp.h> OPAMP configuraion, including mode, internal reference voltage, positive gain, negative gain and so on.

Public Members

```
bool enable
```

Enable/disable OPAMP.

```
opamp_mode_t mode
```

Opamp mode, available values are kOPAMP_LowNoiseMode and kOPAMP_HighSpeedMode.

```
opamp_bias_current_trim_option_t trimOption
```

Bias current trim option, please refer to opamp_bias_current_trim_option_t.

```
opamp_internal_ref_voltage_t intRefVoltage
```

Internal reference voltage, please refer to opamp_internal_ref_voltage_t.

```
opamp_positive_ref_voltage_t posRefVoltage
```

Positive part reference voltage, please refer to opamp_positive_ref_voltage_t.

```
opamp_positive_gain_t posGain
```

Positive part programmable gain, please refer to opamp_positive_gain_t.

```
opamp_negative_gain_t negGain
```

Negative part programmable gain, please refer to opamp_negative_gain_t.

```
bool enablePosADCsw
```

Positive part reference voltage switch to ADC channel or not.

- **true** Positive part reference voltage switch to ADC channel.
- **false** Positive part reference voltage do not switch to ADC channel.

2.66 OSTIMER: OS Event Timer Driver

```
void OSTIMER_Init(OSTIMER_Type *base)
```

Initializes an OSTIMER by turning its bus clock on.

`void OSTIMER_Deinit(OSTIMER_Type *base)`

Deinitializes a OSTIMER instance.

This function shuts down OSTIMER bus clock

Parameters

- `base` – OSTIMER peripheral base address.

`uint64_t OSTIMER_GrayToDecimal(uint64_t gray)`

Translate the value from gray-code to decimal.

Parameters

- `gray` – The gray value input.

Returns

The decimal value.

`static inline uint64_t OSTIMER_DecimalToGray(uint64_t dec)`

Translate the value from decimal to gray-code.

Parameters

- `dec` – The decimal value.

Returns

The gray code of the input value.

`uint32_t OSTIMER_GetStatusFlags(OSTIMER_Type *base)`

Get OSTIMER status Flags.

This returns the status flag. Currently, only match interrupt flag can be got.

Parameters

- `base` – OSTIMER peripheral base address.

Returns

status register value

`void OSTIMER_ClearStatusFlags(OSTIMER_Type *base, uint32_t mask)`

Clear Status Interrupt Flags.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

- `base` – OSTIMER peripheral base address.
- `mask` – Clear bit mask.

Returns

none

`status_t OSTIMER_SetMatchRawValue(OSTIMER_Type *base, uint64_t count, ostimer_callback_t cb)`

Set the match raw value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format may be gray-code, if so, please using `OSTIMER_SetMatchValue()`.

Parameters

- `base` – OSTIMER peripheral base address.
- `count` – OSTIMER timer match value.(Value may be gray-code format)

- `cb` – OSTIMER callback (can be left as NULL if none, otherwise should be a `void func(void)`).

Return values

`kStatus_Success` – - Set match raw value and enable interrupt Successfully.

`status_t` OSTIMER_SetMatchValue(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

Parameters

- `base` – OSTIMER peripheral base address.
- `count` – OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code in API if the IP counter is gray encoded.)
- `cb` – OSTIMER callback (can be left as NULL if none, otherwise should be a `void func(void)`).

Return values

`kStatus_Success` – - Set match raw value and enable interrupt Successfully.

static inline void OSTIMER_SetMatchRegister(OSTIMER_Type *base, uint64_t value)

Set value to OSTIMER MATCH register directly.

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER_DecimalToGray could convert decimal value to gray code.

Parameters

- `base` – OSTIMER peripheral base address.
- `value` – OSTIMER timer match value (Value is gray-code format).

static inline uint64_t OSTIMER_GetMatchRegister(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get the match value from OSTIMER. The value of timer match is gray code format.

Parameters

- `base` – OSTIMER peripheral base address.

Returns

Value of match register, data format is gray code.

static inline uint64_t OSTIMER_GetMatchValue(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get a match value from OSTIMER.

Parameters

- `base` – OSTIMER peripheral base address.

Returns

Value of match register.

static inline void OSTIMER_EnableMatchInterrupt(OSTIMER_Type *base)

Enable the OSTIMER counter match interrupt.

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline void OSTIMER_DisableMatchInterrupt(OSTIMER_Type *base)

Disable the OSTIMER counter match interrupt.

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline uint64_t OSTIMER_GetCurrentTimerRawValue(OSTIMER_Type *base)

Get current timer raw count value from OSTIMER.

This function will get the timer count value from OS timer register. The raw value of timer count may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, may be gray code format.

uint64_t OSTIMER_GetCurrentTimerValue(OSTIMER_Type *base)

Get current timer count value from OSTIMER.

This function will get a decimal timer count value. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formatted to decimal value.

static inline uint64_t OSTIMER_GetCaptureRawValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a captured value from OSTIMER. The Raw value of timer capture may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of capture register, data format may be gray code.

uint64_t OSTIMER_GetCaptureValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a capture decimal-value from OSTIMER. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of capture register, data format is decimal.

void OSTIMER_HandleIRQ(OSTIMER_Type *base, *ostimer_callback_t* cb)

OS timer interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – OS timer peripheral base address.
- cb – callback scheduled for this instance of OS timer

Returns

none

FSL_OSTIMER_DRIVER_VERSION

OSTIMER driver version.

enum *_ostimer_flags*

OSTIMER status flags.

Values:

enumerator kOSTIMER_MatchInterruptFlag

Match interrupt flag bit, sets if the match value was reached.

typedef void (**ostimer_callback_t*)(void)

ostimer callback function.

2.67 PINT: Pin Interrupt and Pattern Match Driver

FSL_PINT_DRIVER_VERSION

enum *_pint_pin_enable*

PINT Pin Interrupt enable type.

Values:

enumerator kPINT_PinIntEnableNone

Do not generate Pin Interrupt

enumerator kPINT_PinIntEnableRiseEdge

Generate Pin Interrupt on rising edge

enumerator kPINT_PinIntEnableFallEdge

Generate Pin Interrupt on falling edge

enumerator kPINT_PinIntEnableBothEdges

Generate Pin Interrupt on both edges

enumerator kPINT_PinIntEnableLowLevel

Generate Pin Interrupt on low level

enumerator kPINT_PinIntEnableHighLevel

Generate Pin Interrupt on high level

enum *_pint_int*

PINT Pin Interrupt type.

Values:

enumerator kPINT_PinInt0

Pin Interrupt 0

enumerator kPINT_SecPinInt0

Secure Pin Interrupt 0

enum __pint_pmatch_input_src

PINT Pattern Match bit slice input source type.

Values:

enumerator kPINT_PatternMatchInp0Src

Input source 0

enumerator kPINT_PatternMatchInp1Src

Input source 1

enumerator kPINT_PatternMatchInp2Src

Input source 2

enumerator kPINT_PatternMatchInp3Src

Input source 3

enumerator kPINT_PatternMatchInp4Src

Input source 4

enumerator kPINT_PatternMatchInp5Src

Input source 5

enumerator kPINT_PatternMatchInp6Src

Input source 6

enumerator kPINT_PatternMatchInp7Src

Input source 7

enumerator kPINT_SecPatternMatchInp0Src

Input source 0

enumerator kPINT_SecPatternMatchInp1Src

Input source 1

enum __pint_pmatch_bslic

PINT Pattern Match bit slice type.

Values:

enumerator kPINT_PatternMatchBSlice0

Bit slice 0

enumerator kPINT_SecPatternMatchBSlice0

Bit slice 0

enum __pint_pmatch_bslic_cfg

PINT Pattern Match configuration type.

Values:

enumerator kPINT_PatternMatchAlways

Always Contributes to product term match

enumerator kPINT_PatternMatchStickyRise

Sticky Rising edge

```

enumerator kPINT_PatternMatchStickyFall
    Sticky Falling edge
enumerator kPINT_PatternMatchStickyBothEdges
    Sticky Rising or Falling edge
enumerator kPINT_PatternMatchHigh
    High level
enumerator kPINT_PatternMatchLow
    Low level
enumerator kPINT_PatternMatchNever
    Never contributes to product term match
enumerator kPINT_PatternMatchBothEdges
    Either rising or falling edge
typedef enum _pint_pin_enable pint_pin_enable_t
    PINT Pin Interrupt enable type.
typedef enum _pint_int pint_pin_int_t
    PINT Pin Interrupt type.
typedef enum _pint_pmatch_input_src pint_pmatch_input_src_t
    PINT Pattern Match bit slice input source type.
typedef enum _pint_pmatch_bslice pint_pmatch_bslice_t
    PINT Pattern Match bit slice type.
typedef enum _pint_pmatch_bslice_cfg pint_pmatch_bslice_cfg_t
    PINT Pattern Match configuration type.
typedef struct _pint_status pint_status_t
    PINT event status.
typedef void (*pint_cb_t)(pint_pin_int_t pintr, pint_status_t *status)
    PINT Callback function.
typedef struct _pint_pmatch_cfg pint_pmatch_cfg_t
void PINT_Init(PINT_Type *base)
    Initialize PINT peripheral.
    This function initializes the PINT peripheral and enables the clock.

```

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

```

void PINT_SetCallback(PINT_Type *base, pint_cb_t callback)
    Set PINT callback.
    This function set the callback for PINT interrupt handler.

```

Parameters

- base – Base address of the PINT peripheral.
- callback – Callback.

Return values

None. –

```
void PINT_PinInterruptConfig(PINT_Type *base, pint_pin_int_t intr, pint_pin_enable_t enable)
```

Configure PINT peripheral pin interrupt.

This function configures a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- intr – Pin interrupt.
- enable – Selects detection logic.

Return values

None. –

```
void PINT_PinInterruptGetConfig(PINT_Type *base, pint_pin_int_t pintr, pint_pin_enable_t *enable)
```

Get PINT peripheral pin interrupt configuration.

This function returns the configuration of a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.
- enable – Pointer to store the detection logic.

Return values

None. –

```
void PINT_PinInterruptClrStatus(PINT_Type *base, pint_pin_int_t pintr)
```

Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.

This function clears the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

None. –

```
static inline uint32_t PINT_PinInterruptGetStatus(PINT_Type *base, pint_pin_int_t pintr)
```

Get Selected pin interrupt status.

This function returns the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

status – = 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

```
void PINT_PinInterruptClrStatusAll(PINT_Type *base)
```

Clear all pin interrupts status only when pins were triggered by edge-sensitive.

This function clears the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

```
static inline uint32_t PINT_PinInterruptGetStatusAll(PINT_Type *base)
```

Get all pin interrupts status.

This function returns the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

status – Each bit position indicates the status of corresponding pin interrupt.
= 0 No pin interrupt request. = 1 Pin interrupt request active.

```
static inline void PINT_PinInterruptClrFallFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Clear Selected pin interrupt fall flag.

This function clears the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

None. –

```
static inline uint32_t PINT_PinInterruptGetFallFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Get selected pin interrupt fall flag.

This function returns the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

flag – = 0 Falling edge has not been detected. = 1 Falling edge has been detected.

```
static inline void PINT_PinInterruptClrFallFlagAll(PINT_Type *base)
```

Clear all pin interrupt fall flags.

This function clears the fall flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

```
static inline uint32_t PINT_PinInterruptGetFallFlagAll(PINT_Type *base)
```

Get all pin interrupt fall flags.

This function returns the fall flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

flags – Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

```
static inline void PINT_PinInterruptClrRiseFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Clear Selected pin interrupt rise flag.

This function clears the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

None. –

```
static inline uint32_t PINT_PinInterruptGetRiseFlag(PINT_Type *base, pint_pin_int_t pintr)
```

Get selected pin interrupt rise flag.

This function returns the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

flag – = 0 Rising edge has not been detected. = 1 Rising edge has been detected.

```
static inline void PINT_PinInterruptClrRiseFlagAll(PINT_Type *base)
```

Clear all pin interrupt rise flags.

This function clears the rise flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

```
static inline uint32_t PINT_PinInterruptGetRiseFlagAll(PINT_Type *base)
```

Get all pin interrupt rise flags.

This function returns the rise flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

flags – Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

```
void PINT_PatternMatchConfig(PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
```

Configure PINT pattern match.

This function configures a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

```
void PINT_PatternMatchGetConfig(PINT_Type *base, pint_pmatch_bslice_t bslice,  
                                pint_pmatch_cfg_t *cfg)
```

Get PINT pattern match configuration.

This function returns the configuration of a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

```
static inline uint32_t PINT_PatternMatchGetStatus(PINT_Type *base, pint_pmatch_bslice_t  
                                                  bslice)
```

Get pattern match bit slice status.

This function returns the status of selected bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.

Return values

status – = 0 Match has not been detected. = 1 Match has been detected.

```
static inline uint32_t PINT_PatternMatchGetStatusAll(PINT_Type *base)
```

Get status of all pattern match bit slices.

This function returns the status of all bit slices.

Parameters

- base – Base address of the PINT peripheral.

Return values

status – Each bit position indicates the match status of corresponding bit slice.
= 0 Match has not been detected. = 1 Match has been detected.

```
uint32_t PINT_PatternMatchResetDetectLogic(PINT_Type *base)
```

Reset pattern match detection logic.

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

- base – Base address of the PINT peripheral.

Return values

pmstatus – Each bit position indicates the match status of corresponding bit slice.
= 0 Match was detected. = 1 Match was not detected.

```
static inline void PINT_PatternMatchEnable(PINT_Type *base)
```

Enable pattern match function.

This function enables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchDisable(PINT_Type *base)

Disable pattern match function.

This function disables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchEnableRXEV(PINT_Type *base)

Enable RXEV output.

This function enables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchDisableRXEV(PINT_Type *base)

Disable RXEV output.

This function disables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

void PINT_EnableCallback(PINT_Type *base)

Enable callback.

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

void PINT_DisableCallback(PINT_Type *base)

Disable callback.

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

- base – Base address of the peripheral.

Return values

None. –

void PINT_Deinit(PINT_Type *base)

Deinitialize PINT peripheral.

This function disables the PINT clock.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

`void PINT_EnableCallbackByIndex(PINT_Type *base, pint_pin_int_t pintIdx)`
enable callback by pin index.

This function enables callback by pin index instead of enabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values

None. –

`void PINT_DisableCallbackByIndex(PINT_Type *base, pint_pin_int_t pintIdx)`
disable callback by pin index.

This function disables callback by pin index instead of disabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values

None. –

`PINT_USE_LEGACY_CALLBACK`

`PININT_BITSLICE_SRC_START`

`PININT_BITSLICE_SRC_MASK`

`PININT_BITSLICE_CFG_START`

`PININT_BITSLICE_CFG_MASK`

`PININT_BITSLICE_ENDP_MASK`

`PINT_PIN_INT_LEVEL`

`PINT_PIN_INT_EDGE`

`PINT_PIN_INT_FALL_OR_HIGH_LEVEL`

`PINT_PIN_INT_RISE`

`PINT_PIN_RISE_EDGE`

`PINT_PIN_FALL_EDGE`

`PINT_PIN_BOTH_EDGE`

`PINT_PIN_LOW_LEVEL`

`PINT_PIN_HIGH_LEVEL`

`struct __pint_status`

#include <fsl_pint.h> PINT event status.

`struct __pint_pmatch_cfg`

#include <fsl_pint.h>

2.68 PLU: Programmable Logic Unit

void PLU_Init(PLU_Type *base)

Enable the PLU clock and reset the module.

Note: This API should be called at the beginning of the application using the PLU driver.

Parameters

- base – PLU peripheral base address

void PLU_Deinit(PLU_Type *base)

Gate the PLU clock.

Parameters

- base – PLU peripheral base address

static inline void PLU_SetLutInputSource(PLU_Type *base, *plu_lut_index_t* lutIndex,
plu_lut_in_index_t lutInIndex, *plu_lut_input_source_t*
inputSrc)

Set Input source of LUT.

Note: An external clock must be applied to the PLU_CLKIN input when using FFs. For each LUT, the slot associated with the output from LUTn itself is tied low.

Parameters

- base – PLU peripheral base address.
- lutIndex – LUT index (see *plu_lut_index_t* typedef enumeration).
- lutInIndex – LUT input index (see *plu_lut_in_index_t* typedef enumeration).
- inputSrc – LUT input source (see *plu_lut_input_source_t* typedef enumeration).

static inline void PLU_SetOutputSource(PLU_Type *base, *plu_output_index_t* outputIndex,
plu_output_source_t outputSrc)

Set Output source of PLU.

Note: An external clock must be applied to the PLU_CLKIN input when using FFs.

Parameters

- base – PLU peripheral base address.
- outputIndex – PLU output index (see *plu_output_index_t* typedef enumeration).
- outputSrc – PLU output source (see *plu_output_source_t* typedef enumeration).

static inline void PLU_SetLutTruthTable(PLU_Type *base, *plu_lut_index_t* lutIndex, uint32_t
truthTable)

Set Truth Table of LUT.

Parameters

- base – PLU peripheral base address.
- lutIndex – LUT index (see *plu_lut_index_t* typedef enumeration).
- truthTable – Truth Table value.

```
static inline uint32_t PLU_ReadOutputState(PLU_Type *base)
```

Read the current state of the 8 designated PLU Outputs.

Note: The PLU bus clock must be re-enabled prior to reading the Output Register if PLU bus clock is shut-off.

Parameters

- base – PLU peripheral base address.

Returns

Current PLU output state value.

```
FSL_PLU_DRIVER_VERSION
```

Version 2.2.1

```
enum __plu_lut_index
```

Index of LUT.

Values:

```
enumerator kPLU_LUT_0
```

5-input Look-up Table 0

```
enumerator kPLU_LUT_1
```

5-input Look-up Table 1

```
enumerator kPLU_LUT_2
```

5-input Look-up Table 2

```
enumerator kPLU_LUT_3
```

5-input Look-up Table 3

```
enumerator kPLU_LUT_4
```

5-input Look-up Table 4

```
enumerator kPLU_LUT_5
```

5-input Look-up Table 5

```
enumerator kPLU_LUT_6
```

5-input Look-up Table 6

```
enumerator kPLU_LUT_7
```

5-input Look-up Table 7

```
enumerator kPLU_LUT_8
```

5-input Look-up Table 8

```
enumerator kPLU_LUT_9
```

5-input Look-up Table 9

```
enumerator kPLU_LUT_10
```

5-input Look-up Table 10

```
enumerator kPLU_LUT_11
```

5-input Look-up Table 11

```
enumerator kPLU_LUT_12
```

5-input Look-up Table 12

```
enumerator kPLU_LUT_13
```

5-input Look-up Table 13

enumerator kPLU_LUT_14
5-input Look-up Table 14

enumerator kPLU_LUT_15
5-input Look-up Table 15

enumerator kPLU_LUT_16
5-input Look-up Table 16

enumerator kPLU_LUT_17
5-input Look-up Table 17

enumerator kPLU_LUT_18
5-input Look-up Table 18

enumerator kPLU_LUT_19
5-input Look-up Table 19

enumerator kPLU_LUT_20
5-input Look-up Table 20

enumerator kPLU_LUT_21
5-input Look-up Table 21

enumerator kPLU_LUT_22
5-input Look-up Table 22

enumerator kPLU_LUT_23
5-input Look-up Table 23

enumerator kPLU_LUT_24
5-input Look-up Table 24

enumerator kPLU_LUT_25
5-input Look-up Table 25

enum _plu_lut_in_index

Inputs of LUT. 5 input present for each LUT.

Values:

enumerator kPLU_LUT_IN_0
LUT input 0

enumerator kPLU_LUT_IN_1
LUT input 1

enumerator kPLU_LUT_IN_2
LUT input 2

enumerator kPLU_LUT_IN_3
LUT input 3

enumerator kPLU_LUT_IN_4
LUT input 4

enum _plu_lut_input_source

Available sources of LUT input.

Values:

enumerator kPLU_LUT_IN_SRC_PLU_IN_0
Select PLU input 0 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_1
Select PLU input 1 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_2
Select PLU input 2 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_3
Select PLU input 3 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_4
Select PLU input 4 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_PLU_IN_5
Select PLU input 5 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_0
Select LUT output 0 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_1
Select LUT output 1 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_2
Select LUT output 2 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_3
Select LUT output 3 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_4
Select LUT output 4 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_5
Select LUT output 5 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_6
Select LUT output 6 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_7
Select LUT output 7 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_8
Select LUT output 8 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_9
Select LUT output 9 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_10
Select LUT output 10 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_11
Select LUT output 11 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_12
Select LUT output 12 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_13
Select LUT output 13 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_14
Select LUT output 14 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_15
Select LUT output 15 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_16
Select LUT output 16 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_17
Select LUT output 17 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_18
Select LUT output 18 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_19
Select LUT output 19 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_20
Select LUT output 20 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_21
Select LUT output 21 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_22
Select LUT output 22 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_23
Select LUT output 23 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_24
Select LUT output 24 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_LUT_OUT_25
Select LUT output 25 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_0
Select Flip-Flops state 0 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_1
Select Flip-Flops state 1 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_2
Select Flip-Flops state 2 to be connected to LUTn Input x

enumerator kPLU_LUT_IN_SRC_FLIPFLOP_3
Select Flip-Flops state 3 to be connected to LUTn Input x

enum _plu_output_index

PLU output multiplexer registers.

Values:

enumerator kPLU_OUTPUT_0
PLU OUTPUT 0

enumerator kPLU_OUTPUT_1
PLU OUTPUT 1

enumerator kPLU_OUTPUT_2
PLU OUTPUT 2

enumerator kPLU_OUTPUT_3
PLU OUTPUT 3

enumerator kPLU_OUTPUT_4
PLU OUTPUT 4

enumerator kPLU_OUTPUT_5
PLU OUTPUT 5

enumerator kPLU_OUTPUT_6
PLU OUTPUT 6

enumerator kPLU_OUTPUT_7
PLU OUTPUT 7

enum _plu_output_source

Available sources of PLU output.

Values:

enumerator kPLU_OUT_SRC_LUT_0
Select LUT0 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_1
Select LUT1 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_2
Select LUT2 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_3
Select LUT3 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_4
Select LUT4 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_5
Select LUT5 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_6
Select LUT6 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_7
Select LUT7 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_8
Select LUT8 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_9
Select LUT9 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_10
Select LUT10 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_11
Select LUT11 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_12
Select LUT12 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_13
Select LUT13 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_14
Select LUT14 output to be connected to PLU output

enumerator kPLU_OUT_SRC_LUT_15
Select LUT15 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_16`
Select LUT16 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_17`
Select LUT17 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_18`
Select LUT18 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_19`
Select LUT19 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_20`
Select LUT20 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_21`
Select LUT21 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_22`
Select LUT22 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_23`
Select LUT23 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_24`
Select LUT24 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_LUT_25`
Select LUT25 output to be connected to PLU output

enumerator `kPLU_OUT_SRC_FLIPFLOP_0`
Select Flip-Flops state(0) to be connected to PLU output

enumerator `kPLU_OUT_SRC_FLIPFLOP_1`
Select Flip-Flops state(1) to be connected to PLU output

enumerator `kPLU_OUT_SRC_FLIPFLOP_2`
Select Flip-Flops state(2) to be connected to PLU output

enumerator `kPLU_OUT_SRC_FLIPFLOP_3`
Select Flip-Flops state(3) to be connected to PLU output

typedef enum `_plu_lut_index` `plu_lut_index_t`
Index of LUT.

typedef enum `_plu_lut_in_index` `plu_lut_in_index_t`
Inputs of LUT. 5 input present for each LUT.

typedef enum `_plu_lut_input_source` `plu_lut_input_source_t`
Available sources of LUT input.

typedef enum `_plu_output_index` `plu_output_index_t`
PLU output multiplexer registers.

typedef enum `_plu_output_source` `plu_output_source_t`
Available sources of PLU output.

2.69 Power Driver

enum __power_mode_config

Low Power Modes configuration.

Values:

enumerator kPmu__Sleep

enumerator kPmu__Deep__Sleep

enumerator kPmu__PowerDown

enumerator kPmu__Deep__PowerDown

enum __power_reset_cause

Device Reset Causes.

Values:

enumerator kRESET__CAUSE__POR

Power On Reset

enumerator kRESET__CAUSE__PADRESET

Hardware Pin Reset

enumerator kRESET__CAUSE__BODRESET

Brown-out Detector reset (either BODVBAT or BODCORE)

enumerator kRESET__CAUSE__ARMSYSTEMRESET

ARM System Reset

enumerator kRESET__CAUSE__WDTRRESET

Watchdog Timer Reset

enumerator kRESET__CAUSE__SWRRESET

Software Reset

enumerator kRESET__CAUSE__CDOGRESET

Code Watchdog Reset

enumerator kRESET__CAUSE__DPDRESET__WAKEUPPIO

Any of the 5 wake-up pins

enumerator kRESET__CAUSE__DPDRESET__RTC

Real Time Clock (RTC)

enumerator kRESET__CAUSE__DPDRESET__OSTIMER

OS Event Timer (OSTIMER)

enumerator kRESET__CAUSE__DPDRESET__WAKEUPPIO__RTC

Any of the 5 wake-up pins and RTC (the 2 events occurred within 1 nano-second of each other)

enumerator kRESET__CAUSE__DPDRESET__WAKEUPPIO__OSTIMER

Any of the 5 wake-up pins and OSTIMER (the 2 events occurred within 1 nano-second of each other)

enumerator kRESET__CAUSE__DPDRESET__RTC__OSTIMER

Real Time Clock or OS Event Timer (the 2 events occurred within 1 nano-second of each other)

enumerator kRESET__CAUSE__DPDRESET__WAKEUPPIO__RTC__OSTIMER

Any of the 5 wake-up pins or RTC or OS Event Timer (the 3 events occurred within 1 nano-second of each other)

enumerator kRESET_CAUSE_NOT_RELEVANT

No reset cause (for example, this code is used when waking up from DEEP-SLEEP low power mode)

enumerator kRESET_CAUSE_NOT_DETERMINISTIC

Unknown Reset Cause. Should be treated like “Hardware Pin Reset” from an application point of view.

enum __power_boot_mode

Device Boot Modes.

Values:

enumerator kBOOT_MODE_POWER_UP

All non Low Power Mode wake up (Power On Reset, Pin Reset, BoD Reset, ARM System Reset ...)

enumerator kBOOT_MODE_LP_DEEP_SLEEP

Wake up from DEEP-SLEEP Low Power mode

enumerator kBOOT_MODE_LP_POWER_DOWN

Wake up from POWER-DOWN Low Power mode

enumerator kBOOT_MODE_LP_DEEP_POWER_DOWN

Wake up from DEEP-POWER-DOWN Low Power mode

enum __power_wakeup_pin_t

Device wake up pins events.

Values:

enumerator kWAKEUP_PIN_NONE

No wake up pin event

enumerator kWAKEUP_PIN_0

Wake up pin 0 event

enumerator kWAKEUP_PIN_1

Wake up pin 1 event

enumerator kWAKEUP_PIN_2

Wake up pin 2 event

enumerator kWAKEUP_PIN_3

Wake up pin 3 event

enumerator kWAKEUP_PIN_4

Wake up pin 4 event

enumerator kWAKEUP_PIN_MULTIPLE

More than 1 wake up pins events occurred (within 1 nano-second of each other)

enum __power_pd_bit

analog components power modes control during low power modes

Values:

enumerator kPDRUNCFG_PD_DCDC

enumerator kPDRUNCFG_PD_BIAS

enumerator kPDRUNCFG_PD_BODCORE

enumerator kPDRUNCFG_PD_BODVDDMAIN

enumerator kPDRUNCFG_PD_FRO1M
enumerator kPDRUNCFG_PD_FRO192M
enumerator kPDRUNCFG_PD_FRO32K
enumerator kPDRUNCFG_PD_XTAL32K
enumerator kPDRUNCFG_PD_XTALHF
enumerator kPDRUNCFG_PD_PLL0
enumerator kPDRUNCFG_PD_PLL1
enumerator kPDRUNCFG_PD_USBFSPHY
enumerator kPDRUNCFG_PD_COMP
enumerator kPDRUNCFG_PD_LDOMEM
enumerator kPDRUNCFG_PD_LDOEFUSEPROG
enumerator kPDRUNCFG_PD_LDOXTALHF
enumerator kPDRUNCFG_PD_LDOFLASHNV
enumerator kPDRUNCFG_PD_PLL0_SSCG
enumerator kPDRUNCFG_PD_ROM
enumerator kPDRUNCFG_PD_HSCMP0
enumerator kPDRUNCFG_PD_HSCMP1
enumerator kPDRUNCFG_PD_HSCMP2
enumerator kPDRUNCFG_PD_OPAMP0
enumerator kPDRUNCFG_PD_OPAMP1
enumerator kPDRUNCFG_PD_OPAMP2
enumerator kPDRUNCFG_PD_VREF
enumerator kPDRUNCFG_PD_CMPBIAS
enumerator kPDRUNCFG_PD_HSCMP0_DAC
enumerator kPDRUNCFG_PD_HSCMP1_DAC
enumerator kPDRUNCFG_PD_HSCMP2_DAC
enumerator kPDRUNCFG_PD_DAC0
enumerator kPDRUNCFG_PD_DAC1
enumerator kPDRUNCFG_PD_DAC2
enumerator kPDRUNCFG_STOP_DAC0
enumerator kPDRUNCFG_STOP_DAC1
enumerator kPDRUNCFG_STOP_DAC2
enumerator kPDRUNCFG_ForceUnsigned

enum _power_sram_bit

SRAM instances bit masks.

Values:

enumerator kPOWER_SRAM_RAM_X0
RAM_X0

enumerator kPOWER_SRAM_RAM_00
RAM_00

enumerator kPOWER_SRAM_RAM_01
RAM_01

enumerator kPOWER_SRAM_RAM_02
RAM_02

enumerator kPOWER_SRAM_RAM_03
RAM_03

enumerator kPOWER_SRAM_RAM_10
RAM_10

enumerator kPOWER_SRAM_RAM_20
RAM_20

enumerator kPOWER_SRAM_RAM_30
RAM_30

enumerator kPOWER_SRAM_RAM_40
RAM_40

enumerator kPOWER_SRAM_RAM_41
RAM_41

enumerator kPOWER_SRAM_RAM_42
RAM_42

enumerator kPOWER_SRAM_RAM_43
RAM_43

enumerator kPOWER_SRAM_FLASHCACHE
Reserved. Flash Cache SRAM instance

enumerator kPOWER_SRAM_FLEXSPICACHE
Reserved. FlexSPI Cache SRAM instance

enumerator kPOWER_SRAM_FLEXSPIH2PREG
Reserved. FlexSPI Dual Port Register Files instances

enumerator kPOWER_SRAM_DSLP_MASK
Reserved. DEEP-SLEEP SRAM instances

enumerator kPOWER_SRAM_PDWN_MASK
Reserved. POWER-DOWN SRAM instances

enumerator kPOWER_SRAM_DPWD_MASK
Reserved. DEEP-POWER-DOWN SRAM instances (RAM_20 & RAM_30 excluded).

enumerator kPOWER_SRAM_ForceUnsigned

enum __power_sram_index

SRAM instances indexes.

Values:

enumerator kPOWER_SRAM_IDX_RAM_X0
RAM_X0

enumerator kPOWER_SRAM_IDX_RAM_00
RAM_00

enumerator kPOWER_SRAM_IDX_RAM_01
RAM_01

enumerator kPOWER_SRAM_IDX_RAM_02
RAM_02

enumerator kPOWER_SRAM_IDX_RAM_03
RAM_03

enumerator kPOWER_SRAM_IDX_RAM_10
RAM_10

enumerator kPOWER_SRAM_IDX_RAM_20
RAM_20

enumerator kPOWER_SRAM_IDX_RAM_30
RAM_30

enumerator kPOWER_SRAM_IDX_RAM_40
RAM_40

enumerator kPOWER_SRAM_IDX_RAM_41
RAM_41

enumerator kPOWER_SRAM_IDX_RAM_42
RAM_42

enumerator kPOWER_SRAM_IDX_RAM_43
RAM_43

enumerator kPOWER_SRAM_IDX_FLASHCACHE
Reserved. Flash Cache SRAM instance

enumerator kPOWER_SRAM_IDX_FLEXSPICACHE
Reserved. FlexSPI Cache SRAM instance

enumerator kPOWER_SRAM_IDX_FLEXSPIH2PREG
Reserved. FlexSPI Dual Port Register Files instances

enum __power_sram_pwr_mode

SRAM Power Mode.

Values:

enumerator kPOWER_SRAMPwrActive
Active

enumerator kPOWER_SRAMPwrLightSleep
RESERVED, DO NOT USE (Light Sleep)

enumerator kPOWER_SRAMPwrDeepSleep
Deep Sleep : SRAM content retained

enumerator kPOWER_SRAMPwrShutDown
Shutdown: SRAM content lost

enum __power_bod_vddmain_level
BOD VDDMAIN level.

Values:

enumerator kPOWER_BodVddmainLevel1000mv
VDDMAIN Brown out detector level 1V

enumerator kPOWER_BodVddmainLevel1100mv
VDDMAIN Brown out detector level 1.1V

enumerator kPOWER_BodVddmainLevel1200mv
VDDMAIN Brown out detector level 1.2V

enumerator kPOWER_BodVddmainLevel1300mv
VDDMAIN Brown out detector level 1.3V

enumerator kPOWER_BodVddmainLevel1400mv
VDDMAIN Brown out detector level 1.4V

enumerator kPOWER_BodVddmainLevel1500mv
VDDMAIN Brown out detector level 1.5V

enumerator kPOWER_BodVddmainLevel1600mv
VDDMAIN Brown out detector level 1.6V

enumerator kPOWER_BodVddmainLevel1650mv
VDDMAIN Brown out detector level 1.65V

enumerator kPOWER_BodVddmainLevel1700mv
VDDMAIN Brown out detector level 1.7V

enumerator kPOWER_BodVddmainLevel1750mv
VDDMAIN Brown out detector level 1.75V

enumerator kPOWER_BodVddmainLevel1800mv
VDDMAIN Brown out detector level 1.8V

enumerator kPOWER_BodVddmainLevel1900mv
VDDMAIN Brown out detector level 1.9V

enumerator kPOWER_BodVddmainLevel2000mv
VDDMAIN Brown out detector level 2V

enumerator kPOWER_BodVddmainLevel2100mv
VDDMAIN Brown out detector level 2.1V

enumerator kPOWER_BodVddmainLevel2200mv
VDDMAIN Brown out detector level 2.2V

enumerator kPOWER_BodVddmainLevel2300mv
VDDMAIN Brown out detector level 2.3V

enumerator kPOWER_BodVddmainLevel2400mv
VDDMAIN Brown out detector level 2.4V

enumerator kPOWER_BodVddmainLevel2500mv
VDDMAIN Brown out detector level 2.5V

enumerator kPOWER__BodVddmainLevel2600mv
VDDMAIN Brown out detector level 2.6V

enumerator kPOWER__BodVddmainLevel2700mv
VDDMAIN Brown out detector level 2.7V

enumerator kPOWER__BodVddmainLevel2800mv
VDDMAIN Brown out detector level 2.80 V

enumerator kPOWER__BodVddmainLevel2900mv
VDDMAIN Brown out detector level 2.9V

enumerator kPOWER__BodVddmainLevel3000mv
VDDMAIN Brown out detector level 3.0V

enumerator kPOWER__BodVddmainLevel3100mv
VDDMAIN Brown out detector level 3.1V

enumerator kPOWER__BodVddmainLevel3200mv
VDDMAIN Brown out detector level 3.2V

enumerator kPOWER__BodVddmainLevel3300mv
VDDMAIN Brown out detector level 3.3V

enum __power_bod_core_level
BOD core level.

Values:

enumerator kPOWER__BodCoreLevel0A600mv
core Brown out detector level 600mV for 0A

enumerator kPOWER__BodCoreLevel0A650mv
core Brown out detector level 650mV for 0A

enumerator kPOWER__BodCoreLevel0A700mv
core Brown out detector level 700mV for 0A

enumerator kPOWER__BodCoreLevel0A750mv
core Brown out detector level 750mV for 0A

enumerator kPOWER__BodCoreLevel0A800mv
core Brown out detector level 800mV for 0A

enumerator kPOWER__BodCoreLevel0A850mv
core Brown out detector level 850mV for 0A

enumerator kPOWER__BodCoreLevel0A900mv
core Brown out detector level 900mV for 0A

enumerator kPOWER__BodCoreLevel0A950mv
core Brown out detector level 950mV for 0A

enumerator kPOWER__BodCoreLevel1B929mv
core Brown out detector level 929mV for 1B

enumerator kPOWER__BodCoreLevel1B984mv
core Brown out detector level 984mV for 1B

enumerator kPOWER__BodCoreLevel1B1038mv
core Brown out detector level 1038mV for 1B

enum `__power_bod_hyst`

BODs (VDDMAIN & Core) Hysteresis control.

Values:

enumerator `kPOWER_BodHystLevel25mv`

BOD Hysteresis control level 25mv

enumerator `kPOWER_BodHystLevel50mv`

BOD Hysteresis control level 50mv

enumerator `kPOWER_BodHystLevel75mv`

BOD Hysteresis control level 75mv

enumerator `kPOWER_BodHystLevel100mv`

BOD Hysteresis control level 100mv

enum `__power_core_pwr_source`

Core Power Source.

Values:

enumerator `kPOWER_CoreSrcDCDC`

DCDC

enumerator `kPOWER_CoreSrcLDOCoreHP`

LDO Core High Power Mode

enumerator `kPOWER_CoreSrcLDOCoreLP`

LDO Core Low Power Mode (DO NOT USE : Reserved for test purposes)

enumerator `kPOWER_CoreSrcExternal`

External (DO NOT USE : Reserved for test purposes)

enum `__power_core_pwr_state`

Core Regulators Power State.

Values:

enumerator `kPOWER_CorePwrDisable`

Disable

enumerator `kPOWER_CorePwrEnable`

Enable

enum `__power_status`

Generic Power Library APIs Status codes.

Values:

enumerator `kPOWER_Status_Success`

OK

enumerator `kPOWER_Status_Fail`

Generic error code

typedef enum `__power_mode_config` `power_mode_cfg_t`

Low Power Modes configuration.

typedef enum `__power_reset_cause` `power_reset_cause_t`

Device Reset Causes.

typedef enum `__power_boot_mode` `power_boot_mode_t`

Device Boot Modes.


```
typedef enum _power_wakeup_pin_t power_wakeup_pin_t
```

Device wake up pins events.

```
typedef enum _power_pd_bit pd_bit_t
```

analog components power modes control during low power modes

```
typedef enum _power_sram_bit power_sram_bit_t
```

SRAM instances bit masks.

```
typedef enum _power_sram_index power_sram_index_t
```

SRAM instances indexes.

```
typedef enum _power_sram_pwr_mode power_sram_pwr_mode_t
```

SRAM Power Mode.

```
typedef enum _power_bod_vddmain_level power_bod_vddmain_level_t
```

BOD VDDMAIN level.

```
typedef enum _power_bod_core_level power_bod_core_level_t
```

BOD core level.

```
typedef enum _power_bod_hyst power_bod_hyst_t
```

BODs (VDDMAIN & Core) Hysteresis control.

```
typedef enum _power_core_pwr_source power_core_pwr_source_t
```

Core Power Source.

```
typedef enum _power_core_pwr_state power_core_pwr_state_t
```

Core Regulators Power State.

```
typedef enum _power_status power_status_t
```

Generic Power Library APIs Status codes.

```
static inline void POWER_EnablePD(pd_bit_t en)
```

API to enable PDRUNCFG bit in the Syscon. Note that enabling the bit powers down the peripheral.

Parameters

- `en` – peripheral for which to enable the PDRUNCFG bit

Returns

none

```
static inline void POWER_DisablePD(pd_bit_t en)
```

API to disable PDRUNCFG bit in the Syscon. Note that disabling the bit powers up the peripheral.

Parameters

- `en` – peripheral for which to disable the PDRUNCFG bit

Returns

none

```
power_status_t POWER_PowerInit(void)
```

SoC Power Management Controller initialization.

Returns

`power_status_t`

```
power_status_t POWER_SetCorePowerSource(power_core_pwr_source_t pwr_source)
```

Selects the core logic supply source.

Parameters

- `pwr_source` – : Defines which regulator will be used to power the part core logic (internally)

Returns

`power_status_t`

`power_core_pwr_source_t` `POWER_GetCorePowerSource(void)`

Returns the current core logic supply source.

Returns

`power_core_pwr_source_t`

`power_status_t` `POWER_CorePowerSourceControl(power_core_pwr_source_t pwr_source, power_core_pwr_state_t pwr_state)`

Allows to control the state (enabled or disabled) of the core logic internal regulators (DCDC, LDO_CORE)

Parameters

- `pwr_source` – : Defines which regulator will be enabled or disabled
- `pwr_state` – : Defines the state of the internal regulator indicated by `pwr_source`

Returns

`power_status_t`

`power_status_t` `POWER_SRAMPowerModeControl(power_sram_bit_t sram_inst, power_sram_pwr_mode_t pwr_mode)`

Allows to configure SRAM instances (low) power modes when the part is in ACTIVE mode.

Parameters

- `sram_inst` – : Defines the SRAM instance(s) to be configured.
- `pwr_mode` – : Defines the SRAM low power mode to be applied to all SRAM instances given by `sram_inst`

Returns

`power_status_t`

`power_sram_pwr_mode_t` `POWER_GetSRAMPowerMode(power_sram_index_t sram_index)`

Parameters

- `sram_index` – : SRAM instances indexes

Returns

`power_sram_pwr_mode_t`

`void` `POWER_EnterSleep(void)`

Configures and enters in SLEEP low power mode.

Returns

Nothing

`void` `POWER_EnterDeepSleep(uint32_t exclude_from_pd[2], uint32_t sram_retention_ctrl, uint32_t wakeup_interrupts[4], uint32_t hardware_wake_ctrl)`

Configures and enters in DEEP-SLEEP low power mode.

Parameters

- `exclude_from_pd` – Defines which analog peripherals shall NOT be powered down (it is a 2 x 32-bit vectors, aligned with “`pd_bit_t`” definition)
- `sram_retention_ctrl` – Defines which SRAM instances will be put in “retention” mode during deep-sleep (aligned with “`power_sram_bit_t`” definition)

- `wakeup_interrupts` – Defines which peripheral interrupts can be a wake-up source during deep-sleep (it is a 4 x 32-bit vectors, aligned with “WAKEUP_” definition)
- `hardware_wake_ctrl` – configure DMA services during deep-sleep without waking up entire device (see “LOWPOWER_HWWAKE_” definition).

Returns

Nothing

!!! IMPORTANT NOTES :

- 1 - CPU & System CLock frequency is switched to FRO12MHz and is NOT restored back by the API.

```
void POWER_EnterPowerDown(uint32_t exclude_from_pd[1], uint32_t sram_retention_ctrl,
                          uint32_t wakeup_interrupts[4], uint32_t cpu_retention_addr)
```

Configures and enters in POWERDOWN low power mode.

Parameters

- `exclude_from_pd` – Defines which analog peripherals shall NOT be powered down (it is a 1 x 32-bit vector, aligned with “pd_bit_t” definition)
- `sram_retention_ctrl` – Defines which SRAM instances will be put in “retention” mode during power-down (aligned with “power_sram_bit_t” definition)
- `wakeup_interrupts` – Defines which peripheral interrupts can be a wake-up source during power-down (it is a 2 x 32-bit vectors, aligned with “WAKEUP_” definition)
- `cpu_retention_addr` – Must be:
 - Word aligned (address ending by 0x0, 0x4, 0x8 and 0xC).
 - Between 0x2000_0000 and 0x2000_09FC (inside RAM_00) or
 - Between 0x2000_1000 and 0x2000_19FC (inside RAM_01) or
 - Between 0x2000_2000 and 0x2000_29FC (inside RAM_02) or
 - Between 0x2000_3000 and 0x2000_39FC (inside RAM_03)
 - The CPU state will be stored in SRAM from “cpu_retention_addr” to “cpu_retention_addr + 1540”. Therefore, any data present in this area before calling the function will be lost.

Returns

Nothing

!!! IMPORTANT NOTES :

- 1 - CPU0 & System Clock frequency is switched to FRO12MHz and is NOT restored back by the API.
- 2 - It is the responsibility of the user to make sure that SRAM instance containing the application software stack and variables WILL BE preserved during low power (via parameter “sram_retention_ctrl”)

```
void POWER_EnterDeepPowerDown(uint32_t exclude_from_pd[1], uint32_t sram_retention_ctrl,
                              uint32_t wakeup_interrupts[2], uint32_t wakeup_io_ctrl)
```

Configures and enters in DEEPPowerDown low power mode.

Parameters

- `exclude_from_pd` – Defines which analog peripherals shall NOT be powered down (it is a 1 x 32-bit vector, aligned with “pd_bit_t” definition)

- `sram_retention_ctrl` – Defines which SRAM instances will be put in “retention” mode during deep power-down (aligned with “`power_sram_bit_t`” definition)
- `wakeup_interrupts` – Defines which peripheral interrupts can be a wake-up source during deep power-down (it is a 2 x 32-bit vectors, aligned with “`WAKEUP_`” definition)
- `wakeup_io_ctrl` – configure the 5 wake-up pins that can wake-up the part from deep power-down mode (see “`LOWPOWER_WAKEUPIOSRC_*`” definition)

Returns

Nothing

!!! IMPORTANT NOTES :

- 1 - CPU0 & System Clock frequency is switched to FRO12MHz and is NOT restored back by the API.
- 2 - The HARD FAULT handler should execute from SRAM. (The Hard fault handler should initiate a full chip

`reset`)

`void POWER_SetWakeUpPins(uint32_t wakeup_io_cfg_src, uint32_t wakeup_io_ctrl)`

Configures the 5 wake-up pins to wake up the part in DEEP-SLEEP and POWER-DOWN low power modes.

Parameters

- `wakeup_io_cfg_src` – : for all wake-up pins : indicates if the config is from IOCON (0) or from PMC (1).
- `wakeup_io_ctrl` – the 5 wake-up pins configurations (see “`LOW-POWER_WAKEUPIOSRC_*`” definition)

Returns

Nothing

!!! IMPORTANT NOTES :

- 1 - To be called just before `POWER_EnterDeepSleep()` or `POWER_EnterPowerDown()`.

`void POWER_GetWakeUpCause(power_reset_cause_t *reset_cause, power_boot_mode_t *boot_mode, power_wakeup_pin_t *wakeup_pin_cause)`

Return some key information related to the device reset causes / wake-up sources, for all power modes.

Parameters

- `reset_cause` – : the device reset cause, according to the definition of `power_reset_cause_t` type.
- `boot_mode` – : the device boot mode, according to the definition of `power_boot_mode_t` type.
- `wakeup_pin_cause` – the wake-up pin sources, according to the definition of `power_wakeup_pin_t` type.

Returns

Nothing

`void POWER_SetVoltageForFreq(uint32_t system_freq_hz)`

Configures the device internal power control settings.

Parameters

- `system_freq_hz` – operating frequency required (in Hertz).

Returns

Nothing

prepare on-chip power regulators (DC-DC Converter / Core and Always-on Low Drop-Out regulators) to deliver the amount of power needed for the requested performance level, as defined by the CPU operating frequency.

FSL_POWER_DRIVER_VERSION

power driver version 2.0.1.

WAKEUP_SYS

Low Power Modes Wake up sources.

WAKEUP_SDMA0

[SLEEP, DEEP SLEEP]

WAKEUP_GPIO_GLOBALINT0

[SLEEP, DEEP SLEEP, POWER DOWN]

WAKEUP_GPIO_GLOBALINT1

[SLEEP, DEEP SLEEP, POWER DOWN]

WAKEUP_GPIO_INT0_0

[SLEEP, DEEP SLEEP]

WAKEUP_GPIO_INT0_1

[SLEEP, DEEP SLEEP]

WAKEUP_GPIO_INT0_2

[SLEEP, DEEP SLEEP]

WAKEUP_GPIO_INT0_3

[SLEEP, DEEP SLEEP]

WAKEUP_UTICK

[SLEEP, DEEP SLEEP]

WAKEUP_MRT

[SLEEP,]

WAKEUP_CTIMER0

[SLEEP, DEEP SLEEP]

WAKEUP_CTIMER1

[SLEEP, DEEP SLEEP]

WAKEUP_SCT

[SLEEP,]

WAKEUP_CTIMER3

[SLEEP, DEEP SLEEP]

WAKEUP_FLEXCOMM0

[SLEEP, DEEP SLEEP]

WAKEUP_FLEXCOMM1

[SLEEP, DEEP SLEEP]

WAKEUP_FLEXCOMM2

[SLEEP, DEEP SLEEP]

WAKEUP_FLEXCOMM3

[SLEEP, DEEP SLEEP, POWER DOWN]

WAKEUP_FLEXCOMM4
[SLEEP, DEEP SLEEP]

WAKEUP_FLEXCOMM5
[SLEEP, DEEP SLEEP]

WAKEUP_FLEXCOMM6
[SLEEP, DEEP SLEEP]

WAKEUP_FLEXCOMM7
[SLEEP, DEEP SLEEP]

WAKEUP_ADC0
[SLEEP,]

WAKEUP_ADC1
[SLEEP,]

WAKEUP_ACOMP
[SLEEP, DEEP SLEEP, POWER DOWN]

WAKEUP_DMIC
[SLEEP,]

WAKEUP_HWVAD
[SLEEP, DEEP SLEEP,]

WAKEUP_USB0_NEEDCLK
[SLEEP, DEEP SLEEP]

WAKEUP_USB0
[SLEEP, DEEP SLEEP]

WAKEUP_RTC_ALARM_WAKEUP
[SLEEP, DEEP SLEEP, POWER DOWN, DEEP POWER DOWN]

WAKEUP_EZH_ARCH_B
[SLEEP,]

WAKEUP_WAKEUP_MAILBOX
[SLEEP, DEEP SLEEP,]

WAKEUP_GPIO_INT0_4
[SLEEP, DEEP SLEEP]

WAKEUP_GPIO_INT0_5
[SLEEP, DEEP SLEEP]

WAKEUP_GPIO_INT0_6
[SLEEP, DEEP SLEEP]

WAKEUP_GPIO_INT0_7
[SLEEP, DEEP SLEEP]

WAKEUP_TIMER2
[SLEEP, DEEP SLEEP]

WAKEUP_TIMER4
[SLEEP, DEEP SLEEP]

WAKEUP_OS_EVENT_TIMER
[SLEEP, DEEP SLEEP, POWER DOWN, DEEP POWER DOWN]

WAKEUP_FLEXSPI
[SLEEP,]

WAKEUP_CAN0_0
[SLEEP,]

WAKEUP_CAN0_1
[SLEEP,]

WAKEUP_SPIFILTER
[SLEEP,]

WAKEUP_SEC_HYPERSVISOR_CALL
[SLEEP,]

WAKEUP_SEC_GPIO_INT0_0
[SLEEP, DEEP SLEEP]

WAKEUP_SEC_GPIO_INT0_1
[SLEEP, DEEP SLEEP]

WAKEUP_SEC_VIO
[SLEEP,]

WAKEUP_CSS_IRQ0
[SLEEP,]

WAKEUP_PKC
[SLEEP,]

WAKEUP_PUF
[SLEEP,]

WAKEUP_PQ
[SLEEP,]

WAKEUP_SDMA1
[SLEEP, DEEP SLEEP]

WAKEUP_LSPI_HS
[SLEEP, DEEP SLEEP]

WAKEUP_CODE_WDG
[SLEEP,]

WAKEUP_I3C
[SLEEP, DEEP SLEEP]

WAKEUP_NEUTRON
[SLEEP,]

WAKEUP_CSS_IRQ1
[SLEEP,]

WAKEUP_ANA_GLITCH_DETECT

WAKEUP_DAC0
[SLEEP, DEEP SLEEP]

WAKEUP_DAC1
[SLEEP, DEEP SLEEP]

WAKEUP_DAC2
[SLEEP, DEEP SLEEP]

WAKEUP_HS_COMP0
[SLEEP,]

WAKEUP_HS_COMP1
[SLEEP,]

WAKEUP_HS_COMP2
[SLEEP,]

WAKEUP_FLEXPWM0_CAPTURE
[SLEEP,]

WAKEUP_FLEXPWM0_FAULT
[SLEEP,]

WAKEUP_FLEXPWM0_RELOAD_ERROR
[SLEEP,]

WAKEUP_FLEXPWM0_COMPARE0
[SLEEP,]

WAKEUP_FLEXPWM0_RELOAD0
[SLEEP,]

WAKEUP_FLEXPWM0_COMPARE1
[SLEEP,]

WAKEUP_FLEXPWM0_RELOAD1
[SLEEP,]

WAKEUP_FLEXPWM0_COMPARE2
[SLEEP,]

WAKEUP_FLEXPWM0_RELOAD2
[SLEEP,]

WAKEUP_FLEXPWM0_COMPARE3
[SLEEP,]

WAKEUP_FLEXPWM0_RELOAD3
[SLEEP,]

WAKEUP_FLEXPWM1_CAPTURE
[SLEEP,]

WAKEUP_FLEXPWM1_FAULT
[SLEEP,]

WAKEUP_FLEXPWM1_RELOAD_ERROR
[SLEEP,]

WAKEUP_FLEXPWM1_COMPARE0
[SLEEP,]

WAKEUP_FLEXPWM1_RELOAD0
[SLEEP,]

WAKEUP_FLEXPWM1_COMPARE1
[SLEEP,]

WAKEUP_FLEXPWM1_RELOAD1

[SLEEP,]

WAKEUP_FLEXPWM1_COMPARE2

[SLEEP,]

WAKEUP_FLEXPWM1_RELOAD2

[SLEEP,]

WAKEUP_FLEXPWM1_COMPARE3

[SLEEP,]

WAKEUP_FLEXPWM1_RELOAD3

[SLEEP,]

WAKEUP_ENC0_COMPARE

[SLEEP,]

WAKEUP_ENC0_HOME

[SLEEP,]

WAKEUP_ENC0_WDG

[SLEEP,]

WAKEUP_ENC0_IDX

[SLEEP,]

WAKEUP_ENC1_COMPARE

[SLEEP,]

WAKEUP_ENC1_HOME

[SLEEP,]

WAKEUP_ENC1_WDG

[SLEEP,]

WAKEUP_ENC1_IDX

[SLEEP,]

WAKEUP_ITRC

[SLEEP, DEEP SLEEP, POWER DOWN]

WAKEUP_CF_DSP24L_IRQ0

[SLEEP,]

WAKEUP_CF_DSP24L_IRQ1

[SLEEP,]

WAKEUP_FTM0

[SLEEP,]

LOWPOWER_HWWAKE_FORCED

Sleep Postpone (DEEP-SLEEP)

Force peripheral clocking to stay on during deep-sleep mode.

LOWPOWER_HWWAKE_PERIPHERALS

Wake for Flexcomms. Any Flexcomm FIFO reaching the level specified by its own TXLVL will cause \ peripheral clocking to wake up temporarily while the related status is asserted

LOWPOWER_HWWAKE_DMIC

Wake for DMIC. DMIC being busy will cause peripheral clocking to remain running until DMIC \ completes. Used in conjunction with LOWPOWER_HWWAKE_PERIPHERALS

LOWPOWER_HWWAKE_SDMA0

Wake for DMA0. DMA0 being busy will cause peripheral clocking to remain running until DMA \ completes. Used in conjunction with LOWPOWER_HWWAKE_PERIPHERALS or LOWPOWER_HWWAKE_DAC

LOWPOWER_HWWAKE_SDMA1

Wake for DMA1. DMA0 being busy will cause peripheral clocking to remain running until DMA \ completes. Used in conjunction with LOWPOWER_HWWAKE_PERIPHERALS or LOWPOWER_HWWAKE_DAC

LOWPOWER_HWWAKE_DAC

Wake for DAC0, DAC1, DAC2. Any DAC0/1/2 FIFO reaching the level specified by the configuration \ will generate an asynchronous SDMA0 request, and SDMA0 will wake up the bus \ clock temporarily to transfer data to DAC0/1/2.

LOWPOWER_HWWAKE_ENABLE_FRO192M

Need to be set if FRO192M is disable - via PDCTRL0 - in Deep Sleep mode and any of \ LOWPOWER_HWWAKE_PERIPHERALS, LOWPOWER_HWWAKE_SDMA0, LOWPOWER_HWWAKE_SDMA1 or LOWPOWER_HWWAKE_DAC is \ set

LOWPOWER_CPURETCTRL_ENA_DISABLE

CPU State retention (POWER-DOWN)

In POWER DOWN mode, CPU Retention is disabled

LOWPOWER_CPURETCTRL_ENA_ENABLE

In POWER DOWN mode, CPU Retention is enabled

LOWPOWER_WAKEUPIOSRC_PIO0_INDEX

Wake up I/O sources (DEEP POWER-DOWN)

Pin P1(1)

LOWPOWER_WAKEUPIOSRC_PIO1_INDEX

Pin P0(28)

LOWPOWER_WAKEUPIOSRC_PIO2_INDEX

Pin P1(18)

LOWPOWER_WAKEUPIOSRC_PIO3_INDEX

Pin P1(30)

LOWPOWER_WAKEUPIOSRC_PIO4_INDEX

Pin P0(26)

LOWPOWER_WAKEUPIOSRC_DISABLE

Wake up is disable

LOWPOWER_WAKEUPIOSRC_RISING

Wake up on rising edge

LOWPOWER_WAKEUPIOSRC_FALLING

Wake up on falling edge

LOWPOWER_WAKEUPIOSRC_RISING_FALLING

Wake up on both rising or falling edges

LOWPOWER_WAKEUPIOSRC_PIO0MODE_INDEX

Pin P1(1)

LOWPOWER_WAKEUPIOSRC_PIO1MODE_INDEX

Pin P0(28)

LOWPOWER_WAKEUPIOSRC_PIO2MODE_INDEX

Pin P1(18)

LOWPOWER_WAKEUPIOSRC_PIO3MODE_INDEX

Pin P1(30)

LOWPOWER_WAKEUPIOSRC_PIO4MODE_INDEX

Pin P0(26)

LOWPOWER_WAKEUPIOSRC_IO_MODE_PLAIN

Wake up Pad is plain input

LOWPOWER_WAKEUPIOSRC_IO_MODE_PULLDOWN

Wake up Pad is pull-down

LOWPOWER_WAKEUPIOSRC_IO_MODE_PULLUP

Wake up Pad is pull-up

LOWPOWER_WAKEUPIOSRC_IO_MODE_REPEATER

Wake up Pad is in repeater

LOWPOWER_WAKEUIPIO_PIO0_DISABLEPULLUPDOWN_INDEX

Wake-up I/O 0 pull-up/down disable/enable control index

LOWPOWER_WAKEUIPIO_PIO1_DISABLEPULLUPDOWN_INDEX

Wake-up I/O 1 pull-up/down disable/enable control index

LOWPOWER_WAKEUIPIO_PIO2_DISABLEPULLUPDOWN_INDEX

Wake-up I/O 2 pull-up/down disable/enable control index

LOWPOWER_WAKEUIPIO_PIO3_DISABLEPULLUPDOWN_INDEX

Wake-up I/O 3 pull-up/down disable/enable control index

LOWPOWER_WAKEUIPIO_PIO4_DISABLEPULLUPDOWN_INDEX

Wake-up I/O 4 pull-up/down disable/enable control index

LOWPOWER_WAKEUIPIO_PIO0_DISABLEPULLUPDOWN_MASK

Wake-up I/O 0 pull-up/down disable/enable mask

LOWPOWER_WAKEUIPIO_PIO1_DISABLEPULLUPDOWN_MASK

Wake-up I/O 1 pull-up/down disable/enable mask

LOWPOWER_WAKEUIPIO_PIO2_DISABLEPULLUPDOWN_MASK

Wake-up I/O 2 pull-up/down disable/enable mask

LOWPOWER_WAKEUIPIO_PIO3_DISABLEPULLUPDOWN_MASK

Wake-up I/O 3 pull-up/down disable/enable mask

LOWPOWER_WAKEUIPIO_PIO4_DISABLEPULLUPDOWN_MASK

Wake-up I/O 4 pull-up/down disable/enable mask

LOWPOWER_WAKEUIPIO_PIO0_PULLUPDOWN_INDEX

Wake-up I/O 0 pull-up/down configuration index

LOWPOWER_WAKEUIPIO_PIO1_PULLUPDOWN_INDEX

Wake-up I/O 1 pull-up/down configuration index

LOWPOWER_WAKEUIPIO_PIO2_PULLUPDOWN_INDEX

Wake-up I/O 2 pull-up/down configuration index

LOWPOWER_WAKEUIPIO_PIO3_PULLUPDOWN_INDEX

Wake-up I/O 3 pull-up/down configuration index

LOWPOWER_WAKEUIO_PIO4_PULLUPDOWN_INDEX

Wake-up I/O 4 pull-up/down configuration index

LOWPOWER_WAKEUIO_PIO0_PULLUPDOWN_MASK

Wake-up I/O 0 pull-up/down mask

LOWPOWER_WAKEUIO_PIO1_PULLUPDOWN_MASK

Wake-up I/O 1 pull-up/down mask

LOWPOWER_WAKEUIO_PIO2_PULLUPDOWN_MASK

Wake-up I/O 2 pull-up/down mask

LOWPOWER_WAKEUIO_PIO3_PULLUPDOWN_MASK

Wake-up I/O 3 pull-up/down mask

LOWPOWER_WAKEUIO_PIO4_PULLUPDOWN_MASK

Wake-up I/O 4 pull-up/down mask

LOWPOWER_WAKEUIO_PULLDOWN

Select pull-down

LOWPOWER_WAKEUIO_PULLUP

Select pull-up

LOWPOWER_WAKEUIO_CFG_SRC_IOCON

Wake-up pins configuration (in/out, pull up/down plain input ...) is coming from IOCON (valid for \ DEEP-SLEEP and POWER-DOWN)

LOWPOWER_WAKEUIO_CFG_SRC_PMC

Wake-up pins configuration (in/out, pull up/down plain input ...) is coming from PMC and set up via \ the second parameter (wakeup_io_ctrl) of POWER_SetWakeUpPins API (valid for DEEP-SLEEP and POWER-DOWN)

unsigned int fro_calib_Get_Lib_Ver(void)

void Chip_TIMER_Instance_Freq(CTIMER_Type *base, unsigned int ctimerFreq)

void USB_SOF_Event(void)

FSL_FRO_CALIB_DRIVER_VERSION

FRO_CALIB driver version 2.0.0.

FSL_COMPONENT_ID

2.70 POWERQUAD: PowerQuad hardware accelerator

void PQ_GetDefaultConfig(pq_config_t *config)

Get default configuration.

This function initializes the POWERQUAD configuration structure to a default value. FORMAT register field definitions Bits[15:8] scaler (for scaled 'q31' formats) Bits[5:4] external format. 00b=q15, 01b=q31, 10b=float Bits[1:0] internal format. 00b=q15, 01b=q31, 10b=float POWERQUAD->INAFORMAT = (config->inputAPrescale « 8U) | (config->inputAFormat « 4U) | config->machineFormat

For all Powerquad operations internal format must be float (with the only exception being the FFT related functions, ie FFT/IFFT/DCT/IDCT which must be set to q31). The default values are: config->inputAFormat = kPQ_Float; config->inputAPrescale = 0; config->inputBFormat = kPQ_Float; config->inputBPrescale = 0; config->outputFormat = kPQ_Float; config->outputPrescale = 0; config->tmpFormat = kPQ_Float; config->tmpPrescale = 0; config->machineFormat = kPQ_Float; config->tmpBase = 0xE0000000;

Parameters

- config – Pointer to “pq_config_t” structure.

void PQ_SetConfig(POWERQUAD_Type *base, const *pq_config_t* *config)

Set configuration with format/prescale.

Parameters

- base – POWERQUAD peripheral base address
- config – Pointer to “pq_config_t” structure.

static inline void PQ_SetCoproprocessorScaler(POWERQUAD_Type *base, const *pq_prescale_t* *prescale)

set coprocessor scaler for coprocessor instructions, this function is used to set output saturation and scaling for input/output.

Parameters

- base – POWERQUAD peripheral base address
- prescale – Pointer to “pq_prescale_t” structure.

void PQ_Init(POWERQUAD_Type *base)

Initializes the POWERQUAD module.

Parameters

- base – POWERQUAD peripheral base address.

void PQ_Deinit(POWERQUAD_Type *base)

De-initializes the POWERQUAD module.

Parameters

- base – POWERQUAD peripheral base address.

void PQ_SetFormat(POWERQUAD_Type *base, *pq_computationengine_t* engine, *pq_format_t* format)

Set format for non-coprocessor instructions.

Parameters

- base – POWERQUAD peripheral base address
- engine – Computation engine
- format – Data format

static inline void PQ_WaitDone(POWERQUAD_Type *base)

Wait for the completion.

Parameters

- base – POWERQUAD peripheral base address

static inline void PQ_LnF32(float *pSrc, float *pDst)

Processing function for the floating-point natural log.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_InvF32(float *pSrc, float *pDst)

Processing function for the floating-point reciprocal.

Parameters

- *pSrc – points to the block of input data. The range of the input value is non-zero.
- *pDst – points to the block of output data

static inline void PQ_SqrtF32(float *pSrc, float *pDst)

Processing function for the floating-point square-root.

Parameters

- *pSrc – points to the block of input data. The range of the input value is [0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_InvSqrtF32(float *pSrc, float *pDst)

Processing function for the floating-point inverse square-root.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_EtoxF32(float *pSrc, float *pDst)

Processing function for the floating-point natural exponent.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_EtonxF32(float *pSrc, float *pDst)

Processing function for the floating-point natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_SinF32(float *pSrc, float *pDst)

Processing function for the floating-point sine.

Parameters

- *pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_CosF32(float *pSrc, float *pDst)

Processing function for the floating-point cosine.

Parameters

- *pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_BiquadF32(float *pSrc, float *pDst)

Processing function for the floating-point biquad.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data

static inline void PQ_DivF32(float *x1, float *x2, float *pDst)

Processing function for the floating-point division.

Get x1 / x2.

Parameters

- x1 – x1
- x2 – x2
- *pDst – points to the block of output data

static inline void PQ_Biquad1F32(float *pSrc, float *pDst)

Processing function for the floating-point biquad.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data

static inline int32_t PQ_LnFixed(int32_t val)

Processing function for the fixed natural log.

Parameters

- val – value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns ln(val).

static inline int32_t PQ_InvFixed(int32_t val)

Processing function for the fixed reciprocal.

Parameters

- val – value to be calculated. The range of the input value is non-zero.

Returns

returns inv(val).

static inline uint32_t PQ_SqrtFixed(uint32_t val)

Processing function for the fixed square-root.

Parameters

- val – value to be calculated. The range of the input value is [0 +INFINITY).

Returns

returns sqrt(val).

static inline int32_t PQ_InvSqrtFixed(int32_t val)

Processing function for the fixed inverse square-root.

Parameters

- val – value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns 1/sqrt(val).

static inline int32_t PQ_EtoxFixed(int32_t val)

Processing function for the Fixed natural exponent.

Parameters

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etox^{val} .

static inline int32_t PQ_EtonxFixed(int32_t val)

Processing function for the fixed natural exponent with negative parameter.

Parameters

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etox^{val} .

static inline int32_t PQ_SinQ31(int32_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns

returns $\sin(\text{val})$.

static inline int16_t PQ_SinQ15(int16_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns

returns $\sin(\text{val})$.

static inline int32_t PQ_CosQ31(int32_t val)

Processing function for the fixed cosine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns

returns $\cos(\text{val})$.

static inline int16_t PQ_CosQ15(int16_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns

returns $\sin(\text{val})$.

static inline int32_t PQ_BiquadFixed(int32_t val)

Processing function for the fixed biquad.

Parameters

- val – value to be calculated

Returns

returns biquad(val).

void PQ_VectorLnF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data

- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorLnFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinQ15(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the Q15 vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosQ15(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the Q15 vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinQ31(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosQ31(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorLnFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorBiquadDf2F32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data.

void PQ_VectorBiquadDf2Fixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadDf2Fixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2F32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2Fixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2Fixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

int32_t PQ_ArctanFixed(POWERQUAD_Type *base, int32_t x, int32_t y, pq_cordic_iter_t iteration)

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, the behavior is like c function atan.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the arctan(0.5), the result of PQ_ArctanFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_ArctanFixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is in the range of -2^{26} to 2^{26} , which means $-\pi/2$ to $\pi/2$.

`int32_t PQ_ArctanhFixed(POWERQUAD_Type *base, int32_t x, int32_t y, pq_cordic_iter_t iteration)`

Processing function for the fixed inverse trigonometric.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the `arctanh(0.5)`, the result of `PQ_ArctanhFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24)` is more accurate than `PQ_ArctanhFixed(POWERQUAD, 1, 2, kPQ_Iteration_24)`.

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is radians, 2^{27} means π . The range is -1.118 to 1.118 radians.

`int32_t PQ_Arctan2Fixed(POWERQUAD_Type *base, int32_t x, int32_t y, pq_cordic_iter_t iteration)`

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, it calculates the angle in radians for the quadrant. The behavior is like c function `atan2`.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the `arctan(0.5)`, the result of `PQ_Arctan2Fixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24)` is more accurate than `PQ_Arctan2Fixed(POWERQUAD, 1, 2, kPQ_Iteration_24)`.

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is in the range of -2^{27} to 2^{27} , which means $-\pi$ to π .

```
static inline int32_t PQ_Biquad1Fixed(int32_t val)
```

Processing function for the fixed biquad.

Parameters

- val – value to be calculated

Returns

returns biquad(val).

```
void PQ_TransformCFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the complex FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformRFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the real FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformIFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the inverse complex FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformCDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the complex DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformRDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the real DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_TransformIDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)
```

Processing function for the inverse complex DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

```
void PQ_BiquadBackUpInternalState(POWERQUAD_Type *base, int32_t biquad_num, pq_biquad_state_t *state)
```

Processing function for backup biquad context.

Parameters

- base – POWERQUAD peripheral base address
- biquad_num – biquad side
- state – point to states.

```
void PQ_BiquadRestoreInternalState(POWERQUAD_Type *base, int32_t biquad_num, pq_biquad_state_t *state)
```

Processing function for restore biquad context.

Parameters

- base – POWERQUAD peripheral base address
- biquad_num – biquad side
- state – point to states.

```
void PQ_BiquadCascadeDf2Init(pq_biquad_cascade_df2_instance *S, uint8_t numStages, pq_biquad_state_t *pState)
```

Initialization function for the direct form II Biquad cascade filter.

Parameters

- *S – **[inout]** points to an instance of the filter data structure.
- numStages – **[in]** number of 2nd order stages in the filter.
- *pState – **[in]** points to the state buffer.

```
void PQ_BiquadCascadeDf2F32(const pq_biquad_cascade_df2_instance *S, float *pSrc, float *pDst, uint32_t blockSize)
```

Processing function for the floating-point direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.

- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

```
void PQ_BiquadCascadeDf2Fixed32(const pq_biquad_cascade_df2_instance *S, int32_t *pSrc,  
                                int32_t *pDst, uint32_t blockSize)
```

Processing function for the Q31 direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

```
void PQ_BiquadCascadeDf2Fixed16(const pq_biquad_cascade_df2_instance *S, int16_t *pSrc,  
                                int16_t *pDst, uint32_t blockSize)
```

Processing function for the Q15 direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

```
void PQ_FIR(POWERQUAD_Type *base, const void *pAData, uint16_t ALength, const void  
            *pBData, uint16_t BLength, void *pResult, uint32_t opType)
```

Processing function for the FIR.

Parameters

- base – POWERQUAD peripheral base address
- pAData – the first input sequence
- ALength – number of the first input sequence
- pBData – the second input sequence
- BLength – number of the second input sequence
- pResult – array for the output data
- opType – operation type, could be PQ_FIR_FIR, PQ_FIR_CONVOLUTION, PQ_FIR_CORRELATION.

```
void PQ_FIRIncrement(POWERQUAD_Type *base, uint16_t ALength, uint16_t BLength, int32_t  
                    xOffset)
```

Processing function for the incremental FIR. This function can be used after pq_fir() for incremental FIR operation when new x data are available.

Parameters

- base – POWERQUAD peripheral base address
- ALength – number of input samples
- BLength – number of taps
- xOffset – offset for number of input samples

```
void PQ_MatrixAddition(POWERQUAD_Type *base, uint32_t length, void *pAData, void
                      *pBData, void *pResult)
```

Processing function for the matrix addition.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

```
void PQ_MatrixSubtraction(POWERQUAD_Type *base, uint32_t length, void *pAData, void
                        *pBData, void *pResult)
```

Processing function for the matrix subtraction.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

```
void PQ_MatrixMultiplication(POWERQUAD_Type *base, uint32_t length, void *pAData, void
                          *pBData, void *pResult)
```

Processing function for the matrix multiplication.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

```
void PQ_MatrixProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData,
                    void *pResult)
```

Processing function for the matrix product.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.

- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_VectorDotProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the vector dot product.

Parameters

- base – POWERQUAD peripheral base address
- length – length of vector
- pAData – input vector A
- pBData – input vector B
- pResult – array for the output data.

void PQ_MatrixInversion(POWERQUAD_Type *base, uint32_t length, void *pData, void *pTmpData, void *pResult)

Processing function for the matrix inverse.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pData – input matrix
- pTmpData – input temporary matrix, pTmpData length not less than pData length and 1024 words is sufficient for the largest supported matrix.
- pResult – array for the output data, round down for fixed point.

void PQ_MatrixTranspose(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the matrix transpose.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
- pData – input matrix
- pResult – array for the output data.

void PQ_MatrixScale(POWERQUAD_Type *base, uint32_t length, float misc, const void *pData, void *pResult)

Processing function for the matrix scale.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols

LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.

- misc – scaling parameters
- pData – input matrix
- pResult – array for the output data.

FSL_POWERQUAD_DRIVER_VERSION
Version.

enum pq_computationengine_t
powerquad computation engine

Values:

enumerator kPQ_CP_PQ
Math engine.

enumerator kPQ_CP_MTX
Matrix engine.

enumerator kPQ_CP_FFT
FFT engine.

enumerator kPQ_CP_FIR
FIR engine.

enumerator kPQ_CP_CORDIC
CORDIC engine.

enum pq_format_t
powerquad data structure format type

Values:

enumerator kPQ_16Bit
Int16 Fixed point.

enumerator kPQ_32Bit
Int32 Fixed point.

enumerator kPQ_Float
Float point.

enum pq_cordic_iter_t
CORDIC iteration.

Values:

enumerator kPQ_Iteration_8
Iterate 8 times.

enumerator kPQ_Iteration_16
Iterate 16 times.

enumerator kPQ_Iteration_24
Iterate 24 times.

typedef struct _pq_biquad_param pq_biquad_param_t
Struct to save biquad parameters.

typedef struct _pq_biquad_state pq_biquad_state_t
Struct to save biquad state.

```
typedef union _pq_float pq_float_t
    Conversion between integer and float type.
```

```
PQ__VectorBiquaudDf2F32
```

```
PQ__VectorBiquaudDf2Fixed32
```

```
PQ__VectorBiquaudDf2Fixed16
```

```
PQ__VectorBiquaudCascadeDf2F32
```

```
PQ__VectorBiquaudCascadeDf2Fixed32
```

```
PQ__VectorBiquaudCascadeDf2Fixed16
```

```
PQ__Vector8BiquaudDf2CascadeF32
```

```
PQ__Vector8BiquaudDf2CascadeFixed32
```

```
PQ__Vector8BiquaudDf2CascadeFixed16
```

```
PQ__FLOAT32
```

```
PQ__FIXEDPT
```

```
CP__PQ
```

```
CP__MTX
```

```
CP__FFT
```

```
CP__FIR
```

```
CP__CORDIC
```

```
PQ__TRANS
```

```
PQ__TRIG
```

```
PQ__BIQUAD
```

```
PQ__TRANS_FIXED
```

```
PQ__TRIG_FIXED
```

```
PQ__BIQUAD_FIXED
```

```
PQ__INV
```

```
PQ__LN
```

```
PQ__SQRT
```

```
PQ__INVSQRT
```

```
PQ__ETOX
```

```
PQ__ETONX
```

```
PQ__DIV
```

```
PQ__SIN
```

```
PQ__COS
```

PQ_BIQ0_CALC
PQ_BIQ1_CALC
PQ_COMP0_ONLY
PQ_COMP1_ONLY
CORDIC_ITER(x)
CORDIC_MIU(x)
CORDIC_T(x)
CORDIC_ARCTAN
CORDIC_ARCTANH
INST_BUSY
PQ_ERRSTAT_OVERFLOW
PQ_ERRSTAT_NAN
PQ_ERRSTAT_FIXEDOVERFLOW
PQ_ERRSTAT_UNDERFLOW
PQ_TRANS_CFFT
PQ_TRANS_IFFT
PQ_TRANS_CDCT
PQ_TRANS_IDCT
PQ_TRANS_RFFT
PQ_TRANS_RDCT
PQ_MTX_SCALE
PQ_MTX_MULT
PQ_MTX_ADD
PQ_MTX_INV
PQ_MTX_PROD
PQ_MTX_SUB
PQ_VEC_DOTP
PQ_MTX_TRAN
PQ_FIR_FIR
PQ_FIR_CONVOLUTION
PQ_FIR_CORRELATION
PQ_FIR_INCREMENTAL
_pq_ln0(x)

`_pq_inv0(x)`
`_pq_sqrt0(x)`
`_pq_invsqrt0(x)`
`_pq_etox0(x)`
`_pq_etonx0(x)`
`_pq_sin0(x)`
`_pq_cos0(x)`
`_pq_biquad0(x)`
`_pq_ln_fx0(x)`
`_pq_inv_fx0(x)`
`_pq_sqrt_fx0(x)`
`_pq_invsqrt_fx0(x)`
`_pq_etox_fx0(x)`
`_pq_etonx_fx0(x)`
`_pq_sin_fx0(x)`
`_pq_cos_fx0(x)`
`_pq_biquad0_fx(x)`
`_pq_div0(x)`
`_pq_div1(x)`
`_pq_ln1(x)`
`_pq_inv1(x)`
`_pq_sqrt1(x)`
`_pq_invsqrt1(x)`
`_pq_etox1(x)`
`_pq_etonx1(x)`
`_pq_sin1(x)`
`_pq_cos1(x)`
`_pq_biquad1(x)`
`_pq_ln_fx1(x)`
`_pq_inv_fx1(x)`
`_pq_sqrt_fx1(x)`
`_pq_invsqrt_fx1(x)`
`_pq_etox_fx1(x)`

`_pq_etonx_fx1(x)`

`_pq_sin_fx1(x)`

`_pq_cos_fx1(x)`

`_pq_biquad1_fx(x)`

`_pq_readMult0()`

`_pq_readAdd0()`

`_pq_readMult1()`

`_pq_readAdd1()`

`_pq_readMult0_fx()`

`_pq_readAdd0_fx()`

`_pq_readMult1_fx()`

`_pq_readAdd1_fx()`

`PQ_LN_INF`

Parameter used for vector $\ln(x)$

`PQ_INV_INF`

Parameter used for vector $1/x$

`PQ_SQRT_INF`

Parameter used for vector $\sqrt{x}$

`PQ_ISQRT_INF`

Parameter used for vector $1/\sqrt{x}$

`PQ_ETOX_INF`

Parameter used for vector e^x

`PQ_ETONX_INF`

Parameter used for vector e^{-x}

`PQ_SIN_INF`

Parameter used for vector $\sin(x)$

`PQ_COS_INF`

Parameter used for vector $\cos(x)$

`PQ_RUN_OPCODE_R3_R2(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R5_R4(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R7_R6(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_Vector8_FP(middle, last, BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R2_R3(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R4_R5(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_RUN_OPCODE_R6_R7(BATCH_OPCODE, BATCH_MACHINE)`

`PQ_Vector8_FX(middle, last, BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)`

PQ_Initiate_Vector_Func(pSrc, pDst)

Start 32-bit data vector calculation.

Start the vector calculation, the input data could be float, int32_t or Q31.

Parameters

- pSrc – Pointer to the source data.
- pDst – Pointer to the destination data.

PQ_End_Vector_Func()

End vector calculation.

This function should be called after vector calculation.

PQ_StartVector(PSRC, PDST, LENGTH)

Start 32-bit data vector calculation.

Start the vector calculation, the input data could be float, int32_t or Q31.

Parameters

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ_StartVectorFixed16(PSRC, PDST, LENGTH)

Start 16-bit data vector calculation.

Start the vector calculation, the input data could be int16_t. This function should be use with PQ_Vector8Fixed16.

Parameters

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ_StartVectorQ15(PSRC, PDST, LENGTH)

Start Q15-bit data vector calculation.

Start the vector calculation, the input data could be Q15. This function should be use with PQ_Vector8Q15. This function is dedicate for SinQ15/CosQ15 vector calculation. Because PowerQuad only supports Q31 Sin/Cos fixed function, so the input Q15 data is left shift 16 bits first, after Q31 calculation, the output data is right shift 16 bits.

Parameters

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ_EndVector()

End vector calculation.

This function should be called after vector calculation.

PQ_Vector8F32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Float data vector calculation.

Float data vector calculation, the input data should be float. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Fixed32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 32-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF, PQ_SIN_INF, PQ_COS_INF. When this function is used for sin/cos calculation, the input data should be in the format Q1.31. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int32_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Fixed16(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 16-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int16_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Q15(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Q15 data vector calculation.

Q15 data vector calculation, this function should only be used for sin/cos Q15 calculation, and the coprocessor output prescaler must be set to 31 before this function. This function loads Q15 data and left shift 16 bits, calculate and right shift 16 bits, then stores to the output array. The input range -1 to 1 means -pi to pi. For example, to calculate sin of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {...}
int16_t output[VECTOR_LEN];
const pq_prescale_t prescale =
{
    .inputPrescale = 0,
    .outputPrescale = 31,
    .outputSaturate = 0
};

PQ_SetCoprocessorScaler(POWERQUAD, const pq_prescale_t *prescale);

PQ_StartVectorQ15(pSrc, pDst, length);
```

(continues on next page)

(continued from previous page)

```
PQ_Vector8Q15(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_DF2_Vector8_FP(middle, last)

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FP(false,false);
PQ_DF2_Vector8_FP(true,true);
PQ_End_Vector_Func();
```

PQ_DF2_Vector8_FX(middle, last)

Fixed data vector biquad direct form II calculation.

Biquad filter, the input and output data are fixed data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FX(false,false);
PQ_DF2_Vector8_FX(true,true);
PQ_End_Vector_Func();
```

PQ_Vector8BiquadDf2F32()

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```

#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2F32();
PQ_EndVector();

```

PQ_Vector8BiquadDf2Fixed32()

Fixed 32-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q31 or 32-bit integer. Biquad side 0 is used.
Example:

```

#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed32();
PQ_EndVector();

```

PQ_Vector8BiquadDf2Fixed16()

Fixed 16-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q15 or 16-bit integer. Biquad side 0 is used.
Example:

```

#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int16_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,

```

(continues on next page)

(continued from previous page)

```

        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed16();
PQ_EndVector();

```

PQ_DF2_Cascade_Vector8_FP(middle, last)

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FP(false, false);
PQ_DF2_Cascade_Vector8_FP(true, true);
PQ_End_Vector_Func();

```

PQ_DF2_Cascade_Vector8_FX(middle, last)

Fixed data vector direct form II biquad cascade filter.

The input and output data are fixed data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FX(false, false);
PQ_DF2_Cascade_Vector8_FX(true, true);
PQ_End_Vector_Func();

```

PQ_Vector8BiquadDf2CascadeF32()

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
    },
};

```

(continues on next page)

(continued from previous page)

```

        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeF32();
PQ_EndVector();

```

PQ_Vector8BiquadDf2CascadeFixed32()

Fixed 32-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 32-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed32();
PQ_EndVector();

```

PQ_Vector8BiquadDf2CascadeFixed16()

Fixed 16-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 16-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};

```

(continues on next page)

(continued from previous page)

```

int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed16();
PQ_EndVector();

```

POWERQUAD_MAKE_MATRIX_LEN(mat1Row, mat1Col, mat2Col)

Make the length used for matrix functions.

PQ_Q31_2_FLOAT(x)

Convert Q31 to float.

PQ_Q15_2_FLOAT(x)

Convert Q15 to float.

struct pq_prescale_t

#include <fsl_powerquad.h> Coprocessor prescale.

Public Members

int8_t inputPrescale

Input prescale.

int8_t outputPrescale

Output prescale.

int8_t outputSaturate

Output saturate at n bits, for example 0x11 is 8 bit space, the value will be truncated at +127 or -128.

struct pq_config_t

#include <fsl_powerquad.h> powerquad data structure format

Public Members*pq_format_t* inputAFormat

Input A format.

int8_t inputAPrescaleInput A prescale, for example 1.5 can be $1.5 \cdot 2^n$ if you scale by 'shifting' ('scaling' by a factor of n).*pq_format_t* inputBFormat

Input B format.

int8_t inputBPrescale

Input B prescale.

pq_format_t outputFormat

Out format.

int8_t outputPrescale

Out prescale.

pq_format_t tmpFormat

Temp format.

int8_t tmpPrescale

Temp prescale.

pq_format_t machineFormat

Machine format.

uint32_t *tmpBase

Tmp base address.

struct __pq_biquad_param

#include <fsl_powerquad.h> Struct to save biquad parameters.

Public Members*float* v_n_1

v[n-1], set to 0 when initialization.

float v_n

v[n], set to 0 when initialization.

float a_1

a[1]

float a_2

a[2]

float b_0

b[0]

float b_1

b[1]

float b_2

b[2]

struct __pq_biquad_state

#include <fsl_powerquad.h> Struct to save biquad state.

Public Members

pq_biquad_param_t param

Filter parameter.

uint32_t compreg

Internal register, set to 0 when initialization.

struct pq_biquad_cascade_df2_instance

#include <fsl_powerquad.h> Instance structure for the direct form II Biquad cascade filter.

Public Members

uint8_t numStages

Number of 2nd order stages in the filter.

pq_biquad_state_t *pState

Points to the array of state coefficients.

union __pq_float

#include <fsl_powerquad.h> Conversion between integer and float type.

Public Members

float floatX

Float type.

uint32_t integerX

Unsigned interger type.

2.71 PRINCE: PRINCE bus crypto engine

FSL_PRINCE_DRIVER_VERSION

PRINCE driver version 2.6.0.

Current version: 2.6.0

Change log:

- Version 2.0.0
 - Initial version.
- Version 2.1.0
 - Update for the A1 rev. of LPC55Sxx serie.
- Version 2.2.0
 - Add runtime checking of the A0 and A1 rev. of LPC55Sxx serie to support both silicone revisions.
- Version 2.3.0
 - Add support for LPC55S1x and LPC55S2x series
- Version 2.3.0
 - Fix MISRA-2012 issues.
- Version 2.3.1
 - Add support for LPC55S0x series

- Version 2.3.2
 - Fix documentation of enumeration. Extend PRINCE example.
- Version 2.4.0
 - Add support for LPC55S3x series
- Version 2.5.0
 - Add PRINCE_Config() and PRINCE_Reconfig() features.
- Version 2.5.1
 - Fix build error due to renamed symbols
- Version 2.6.0
 - Renamed CSS to ELS

enum _skboot_status

Secure status enumeration.

Values:

enumerator kStatus_SKBOOT_Success

PRINCE Success

enumerator kStatus_SKBOOT_Fail

PRINCE Fail

enumerator kStatus_SKBOOT_InvalidArgument

PRINCE Invalid argument

enumerator kStatus_SKBOOT_KeyStoreMarkerInvalid

PRINCE Invalid marker

enum _secure_bool

Secure boolean enumeration.

Values:

enumerator kSECURE_TRUE

PRINCE true

enumerator kSECURE_FALSE

PRINCE false

enum _prince_region

Prince region.

Values:

enumerator kPRINCE_Region0

PRINCE region 0

enumerator kPRINCE_Region1

PRINCE region 1

enumerator kPRINCE_Region2

PRINCE region 2

enum _prince_lock

Prince lock.

Values:

enumerator kPRINCE_Region0Lock
PRINCE region 0 lock

enumerator kPRINCE_Region1Lock
PRINCE region 1 lock

enumerator kPRINCE_Region2Lock
PRINCE region 2 lock

enumerator kPRINCE_MaskLock
PRINCE mask register lock

enum __prince_flags
Prince flag.

Values:

enumerator kPRINCE_Flag_None
PRINCE Flag None

enumerator kPRINCE_Flag_EraseCheck
PRINCE Flag Erase check

enumerator kPRINCE_Flag_WriteCheck
PRINCE Flag Write check

typedef enum _skboot_status skboot_status_t
Secure status enumeration.

typedef enum _secure_bool secure_bool_t
Secure boolean enumeration.

typedef enum _prince_region prince_region_t
Prince region.

typedef enum _prince_lock prince_lock_t
Prince lock.

typedef enum _prince_flags prince_flags_t
Prince flag.

static inline void PRINCE_EncryptEnable(PRINCE_Type *base)
Enable data encryption.

This function enables PRINCE on-the-fly data encryption.

Parameters

- base – PRINCE peripheral address.

static inline void PRINCE_EncryptDisable(PRINCE_Type *base)
Disable data encryption.

This function disables PRINCE on-the-fly data encryption.

Parameters

- base – PRINCE peripheral address.

static inline bool PRINCE_IsEncryptEnable(PRINCE_Type *base)
Is Enable data encryption.

This function test if PRINCE on-the-fly data encryption is enabled.

Parameters

- base – PRINCE peripheral address.

Returns

true if enabled, false if not

static inline void PRINCE_SetMask(PRINCE_Type *base, uint64_t mask)

Sets PRINCE data mask.

This function sets the PRINCE mask that is used to mask decrypted data.

Parameters

- base – PRINCE peripheral address.
- mask – 64-bit data mask value.

static inline void PRINCE_SetLock(PRINCE_Type *base, uint32_t lock)

Locks access for specified region registers or data mask register.

This function sets lock on specified region registers or mask register.

Parameters

- base – PRINCE peripheral address.
- lock – registers to lock. This is a logical OR of members of the enumeration `prince_lock_t`

status_t PRINCE_GenNewIV(*prince_region_t* region, uint8_t *iv_code, bool store, *flash_config_t* *flash_context)

Generate new IV code.

This function generates new IV code and stores it into the persistent memory. Ensure about 800 bytes free space on the stack when calling this routine with the store parameter set to true!

Parameters

- region – PRINCE region index.
- iv_code – IV code pointer used for storing the newly generated 52 bytes long IV code.
- store – flag to allow storing the newly generated IV code into the persistent memory (FFR).
- flash_context – pointer to the flash driver context structure.

Returns

kStatus_Success upon success

Returns

kStatus_Fail otherwise, kStatus_Fail is also returned if the key code for the particular PRINCE region is not present in the keystore (though new IV code has been provided)

status_t PRINCE_LoadIV(*prince_region_t* region, uint8_t *iv_code)

Load IV code.

This function enables IV code loading into the PRINCE bus encryption engine.

Parameters

- region – PRINCE region index.
- iv_code – IV code pointer used for passing the IV code.

Returns

kStatus_Success upon success

Returns

kStatus_Fail otherwise

```
status_t PRINCE_SetEncryptForAddressRange(prince_region_t region, uint32_t start_address,  
                                           uint32_t length, flash_config_t *flash_context, bool  
                                           regenerate_iv)
```

Allow encryption/decryption for specified address range.

This function sets the encryption/decryption for specified address range. The SR mask value for the selected Prince region is calculated from provided start_address and length parameters. This calculated value is OR'ed with the actual SR mask value and stored into the PRINCE SR_ENABLE register and also into the persistent memory (FFR) to be used after the device reset. It is possible to define several nonadjacent encrypted areas within one Prince region when calling this function repeatedly. If the length parameter is set to 0, the SR mask value is set to 0 and thus the encryption/decryption for the whole selected Prince region is disabled. Ensure about 800 bytes free space on the stack when calling this routine!

Parameters

- region – PRINCE region index.
- start_address – start address of the area to be encrypted/decrypted.
- length – length of the area to be encrypted/decrypted.
- flash_context – pointer to the flash driver context structure.
- regenerate_iv – flag to allow IV code regenerating, storing into the persistent memory (FFR) and loading into the PRINCE engine

Returns

kStatus_Success upon success

Returns

kStatus_Fail otherwise

```
status_t PRINCE_GetRegionSREnable(PRINCE_Type *base, prince_region_t region, uint32_t  
                                   *sr_enable)
```

Gets the PRINCE Sub-Region Enable register.

This function gets PRINCE SR_ENABLE register.

Parameters

- base – PRINCE peripheral address.
- region – PRINCE region index.
- sr_enable – Sub-Region Enable register pointer.

Returns

kStatus_Success upon success

Returns

kStatus_InvalidArgument

```
status_t PRINCE_GetRegionBaseAddress(PRINCE_Type *base, prince_region_t region, uint32_t  
                                      *region_base_addr)
```

Gets the PRINCE region base address register.

This function gets PRINCE BASE_ADDR register.

Parameters

- base – PRINCE peripheral address.
- region – PRINCE region index.
- region_base_addr – Region base address pointer.

Returns

kStatus_Success upon success

Returns

kStatus_InvalidArgument

status_t PRINCE_SetRegionIV(PRINCE_Type *base, *prince_region_t* region, const uint8_t iv[8])

Sets the PRINCE region IV.

This function sets specified AES IV for the given region.

Parameters

- base – PRINCE peripheral address.
- region – Selection of the PRINCE region to be configured.
- iv – 64-bit AES IV in little-endian byte order.

status_t PRINCE_SetRegionBaseAddress(PRINCE_Type *base, *prince_region_t* region, uint32_t region_base_addr)

Sets the PRINCE region base address.

This function configures PRINCE region base address.

Parameters

- base – PRINCE peripheral address.
- region – Selection of the PRINCE region to be configured.
- region_base_addr – Base Address for region.

status_t PRINCE_SetRegionSREnable(PRINCE_Type *base, *prince_region_t* region, uint32_t sr_enable)

Sets the PRINCE Sub-Region Enable register.

This function configures PRINCE SR_ENABLE register.

Parameters

- base – PRINCE peripheral address.
- region – Selection of the PRINCE region to be configured.
- sr_enable – Sub-Region Enable register value.

status_t PRINCE_FlashEraseWithChecker(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length. It deals with the flash erase function complementary to the standard erase API of the IAP1 driver. This implementation additionally checks if the whole encrypted PRINCE subregions are erased at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

Parameters

- config – The pointer to the flash driver context structure.
- start – The start address of the desired flash memory to be erased. The start address needs to be prince-sburegion-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be prince-sburegion-size-aligned.
- key – The value used to validate all flash erase APIs.

Returns

kStatus_FLASH_Success API was executed successfully.

Returns

kStatus_FLASH_InvalidArgument An invalid argument is provided.

Returns

kStatus_FLASH_AlignmentError The parameter is not aligned with the specified baseline.

Returns

kStatus_FLASH_AddressError The address is out of range.

Returns

kStatus_FLASH_EraseKeyError The API erase key is invalid.

Returns

kStatus_FLASH_CommandFailure Run-time error during the command execution.

Returns

kStatus_FLASH_CommandNotSupported Flash API is not supported.

Returns

kStatus_FLASH_EccError A correctable or uncorrectable error during command execution.

Returns

kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce Encrypted flash subregions are not erased at once.

status_t PRINCE_FlashProgramWithChecker(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length. It deals with the flash program function complementary to the standard program API of the IAP1 driver. This implementation additionally checks if the whole PRINCE subregions are programmed at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

Parameters

- config – The pointer to the flash driver context structure.
- start – The start address of the desired flash memory to be programmed. Must be prince-sburegion-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be prince-sburegion-size-aligned.

Returns

kStatus_FLASH_Success API was executed successfully.

Returns

kStatus_FLASH_InvalidArgument An invalid argument is provided.

Returns

kStatus_FLASH_AlignmentError Parameter is not aligned with the specified baseline.

Returns

kStatus_FLASH_AddressError Address is out of range.

Returns

kStatus_FLASH_AccessError Invalid instruction codes and out-of bounds addresses.

Returns

kStatus_FLASH_CommandFailure Run-time error during the command execution.

Returns

kStatus_FLASH_CommandFailure Run-time error during the command execution.

Returns

kStatus_FLASH_CommandNotSupported Flash API is not supported.

Returns

kStatus_FLASH_EccError A correctable or uncorrectable error during command execution.

Returns

kStatus_FLASH_SizeError Encrypted flash subregions are not programmed at once.

FSL_PRINCE_DRIVER_SUBREGION_SIZE_IN_KB

FSL_PRINCE_DRIVER_MAX_FLASH_ADDR

ALIGN_DOWN(x, a)

2.72 PUF: Physical Unclonable Function

2.73 Puf_v3_driver

FSL_PUF_V3_DRIVER_VERSION

PUFv3 driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- 2.0.4
 - Fix warning
- 2.0.3
 - Update for various PUF CTRL wrapper
- 2.0.2
 - Fix MISRA issue in driver.
- 2.0.1
 - Fix PUF initialization issue and update driver to reflect SoC header changes.
- 2.0.0
 - Initial version.

Values:

enumerator kStatus_PUF_OperationNotAllowed

```
enumerator kStatus_PUF_AcNotForThisProductPhase1
enumerator kStatus_PUF_AcNotForThisProductPhase2
enumerator kStatus_PUF_AcCorruptedPhase1
enumerator kStatus_PUF_AcCorruptedPhase2
enumerator kStatus_PUF_AcAuthFailedPhase1
enumerator kStatus_PUF_NBOOT_AcAuthFailedPhase2
enumerator kStatus_PUF_QualityVerificationFail
enumerator kStatus_PUF_ContextIncorrect
enumerator kStatus_PUF_DestinationNotAllowed
enumerator kStatus_PUF_Failure
```

```
typedef uint32_t puf_endianness_t
```

```
typedef uint32_t puf_key_dest_t
```

```
typedef uint32_t puf_key_scope_t
```

```
typedef uint32_t puf_result_code_t
```

```
typedef uint32_t puf_sec_level_t
```

```
puf_endianness_t dataEndianness
```

```
uint8_t CKGATING
```

```
puf_key_scope_t keyScopeStarted
```

```
puf_key_scope_t keyScopeEnrolled
```

```
uint32_t userCtx0
```

```
uint32_t userCtx1
```

```
void PUF_GetDefaultConfig(puf_config_t *conf)
```

brief Sets the default configuration of PUF

This function initialize PUF config structure to default values.

Parameters

- *conf* – PUF configuration structure

```
status_t PUF_Init(PUF_Type *base, puf_config_t *conf)
```

brief Initialize PUF

This function enables power to PUF block and waits until the block initializes.

Parameters

- *conf* – PUF configuration structure

Returns

Status of the init operation

`void PUF_Deinit(PUF_Type *base, puf_config_t *conf)`

brief Denitalize PUF

This function disables power to PUF SRAM and peripheral clock.

Parameters

- base – PUF peripheral base address
- conf – PUF configuration structure

`status_t PUF_Enroll(PUF_Type *base, uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)`

brief Enroll PUF

This function derives a digital fingerprint, generates the corresponding Activation Code (AC) and returns it to be stored in an NVM or a file. This step needs to be performed only once for each device. This function may be permanently disallowed by a fuse.

Parameters

- base – PUF peripheral base address
- activationCode – **[out]** Word aligned address of the resulting activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns

Status of enroll operation.

`status_t PUF_Start(PUF_Type *base, const uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)`

brief Start PUF

The Activation Code generated during the Enroll operation is used to reconstruct the digital fingerprint. This needs to be done after every power-up and reset.

Parameters

- base – PUF peripheral base address
- activationCode – **[in]** Word aligned address of the input activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the start operation.
return Status of start operation.

`status_t PUF_Stop(PUF_Type *base)`

brief Stop PUF

The Stop operation removes all key material from PUF flipflops and PUF SRAM, and sets PUF to the Stopped state.

Parameters

- base – PUF peripheral base address

Returns

Status of stop operation.

status_t PUF_GetKey(PUF_Type *base, *puf_key_ctx_t* *keyCtx, *puf_key_dest_t* keyDest, *uint8_t* *key, *size_t* keySize)

brief PUF Get Key

The Get Key operation derives a key from the intrinsic PUF key and externally provided context.

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keyDest – output destination of the derived PUF key
- key – **[out]** Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the derived key in bytes.

Returns

Status of get key operation.

status_t PUF_WrapGeneratedRandom(PUF_Type *base, *puf_key_ctx_t* *keyCtx, *size_t* keySize, *uint8_t* *keyCode, *size_t* keyCodeSize)

brief PUF Wrap generated random

The Wrap Generated Random operation wraps a random key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keySize – Size of the key to be generated in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns

Status of wrap generated random operation.

status_t PUF_Wrap(PUF_Type *base, *puf_key_ctx_t* *keyCtx, *uint8_t* *userKey, *size_t* userKeySize, *uint8_t* *keyCode, *size_t* keyCodeSize)

brief PUF Wrap user key

The Wrap operation wraps a user defined key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct.
- userKey – Word aligned address of input user key.
- userKeySize – Size of the key to be wrapped in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns

Status of wrap operation.

status_t PUF_Unwrap(PUF_Type *base, *puf_key_dest_t* keyDest, uint8_t *keyCode, size_t keyCodeSize, uint8_t *key, size_t keySize)

brief PUF Unwrap user key

The unwrap operation unwraps the key from a previously created Key Code (KC)

Parameters

- base – PUF peripheral base address
- keyDest – output destination of the unwrapped PUF key
- keyCode – **[in]** Word aligned address of the input key code.
- keyCodeSize – Size of the input keycode in bytes.
- key – Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the key to be generated in bytes.

Returns

Status of unwrap operation.

status_t PUF_GenerateRandom(PUF_Type *base, uint8_t *data, size_t size)

brief Generate Random

The Generate Random operation outputs the requested amount of random data as specified in a provided context.

Parameters

- base – PUF peripheral base address
- size – Size of random data to be generated in bytes.

Returns

Status of generate random operation.

status_t PUF_Zeroize(PUF_Type *base)

brief Zeroize PUF

This function clears all PUF internal logic and puts the PUF to zeroized state.

Parameters

- base – PUF peripheral base address

Returns

Status of the zeroize operation.

status_t PUF_Test(PUF_Type *base, uint8_t *score)

brief Test PUF

With the Test PUF operation, diagnostics about the PUF quality is collected and presented in a PUF score.

Parameters

- base – PUF peripheral base address
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns

Status of the test operation.

static inline void PUF_BlockCommand(PUF_Type *base, uint32_t mask)

Blocks specified PUF commands.

This function blocks PUF commands specified by mask parameter.

Parameters

- base – PUF peripheral base address
- mask – Mask of parameters which should be blocked until power-cycle.

Returns

Status of the test operation.

status_t PUF_SetLock(PUF_Type *base, puf_sec_level_t securityLevel)

brief Set lock of PUF operation

Lock the security level of PUF block until key generate, wrap or unwrap operation is completed. Note: Only security level defined in SEC_LOCK register can use PUFv3 or change its security level. Default setting after leaving ROM is Secure-Privilege

Parameters

- base – PUF peripheral base address
- securityLevel – Security level of PUF block.

Returns

Status of the test operation.

status_t PUF_SetCtxMask(PUF_Type *base, uint32_t appCtxMask)

brief Set App Context mask

This function sets Application defined context mask used in conjunction with key user context 2. Whenever bit in this register is 1, corresponding bit in user context 2 provided during key code creation should be zero only.

This register is only modifiable by task running at secure-privilege level.

Parameters

- base – PUF peripheral base address
- appCtxMask – Value of the Application defined context mask.

Returns

Status of the test operation.

kPUF_EndianLittle

kPUF_EndianBig

kPUF_KeyDestRegister

kPUF_KeyDestKeyBus

kPUF_KeyDestInvalid

kPUF_KeyAllowRegister

kPUF_KeyAllowKeyBus

kPUF_KeyAllowAll

kPUF_ResultOK

kPUF_AcNotForThisProductPhase1

kPUF_AcNotForThisProductPhase2

```
kPUF__AcCorruptedPhase1
kPUF__AcCorruptedPhase2
kPUF__AcAuthFailedPhase1
kPUF__AcAuthFailedPhase2
kPUF__QualityVerificationFail
kPUF__ContextIncorrect
kPUF__DestinationNotAllowed
kPUF__Failure
kPUF__NonsecureUser
kPUF__NonsecurePrivilege
kPUF__SecureUser
kPUF__SecurePrivilege
PUF_ACTIVATION_CODE_SIZE
PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x)
SEC_LOCK_PATTERN

struct puf_config_t
    #include <fsl_puf_v3.h>

struct puf_key_ctx_t
    #include <fsl_puf_v3.h>
```

2.74 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_LoadEvery3Oportunity`

Every 3 PWM opportunities

enumerator `kPWM_LoadEvery4Oportunity`

Every 4 PWM opportunities

enumerator `kPWM_LoadEvery5Oportunity`

Every 5 PWM opportunities

enumerator `kPWM_LoadEvery6Oportunity`

Every 6 PWM opportunities

enumerator `kPWM_LoadEvery7Oportunity`

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_DMAResourceDisable

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 writen new value
- `value` – Value that will been 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

2.75 Reset Driver

enum _SYSCON_RSTn

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

Values:

enumerator kROM_RST_SHIFT_RSTn

ROM reset control .

enumerator kSRAM_CTRL1_RST_SHIFT_RSTn

SRAM Controller 1 reset control .

enumerator kSRAM_CTRL2_RST_SHIFT_RSTn

SRAM Controller 2 reset control .

enumerator kSRAM_CTRL3_RST_SHIFT_RSTn

SRAM Controller 3 reset control .

enumerator kSRAM_CTRL4_RST_SHIFT_RSTn

SRAM Controller 4 reset control .

enumerator kFLASH_RST_SHIFT_RSTn

FLASH reset control .

enumerator kFMC_RST_SHIFT_RSTn

FMC reset control .

enumerator kFLEXSPI_RST_SHIFT_RSTn

FLEXSPI reset control .

enumerator kMUX_RST_SHIFT_RSTn

MUX reset control .

enumerator kIOCON_RST_SHIFT_RSTn

IOCON reset control .

enumerator kGPIO0_RST_SHIFT_RSTn

GPIO0 reset control .

enumerator kGPIO1_RST_SHIFT_RSTn

GPIO1 reset control .

enumerator kGPIO2_RST_SHIFT_RSTn
GPIO2 reset control .

enumerator kGPIO3_RST_SHIFT_RSTn
GPIO3 reset control .

enumerator kPINT_RST_SHIFT_RSTn
PINT reset control .

enumerator kGINT_RST_SHIFT_RSTn
GINT reset control .

enumerator kDMA0_RST_SHIFT_RSTn
DMA0 reset control .

enumerator kCRC_RST_SHIFT_RSTn
CRC reset control .

enumerator kWWDT_RST_SHIFT_RSTn
WWDT reset control .

enumerator kRTC_RST_SHIFT_RSTn
RTC reset control .

enumerator kMAILBOX_RST_SHIFT_RSTn
MAILBOX reset control .

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control .

enumerator kADC1_RST_SHIFT_RSTn
ADC1 reset control .

enumerator kDAC0_RST_SHIFT_RSTn
DAC0 reset control .

enumerator kMRT_RST_SHIFT_RSTn
MRT reset control .

enumerator kOSTIMER_RST_SHIFT_RSTn
OSTIMER reset control .

enumerator kSCT_RST_SHIFT_RSTn
SCT reset control .

enumerator kMCAN_RST_SHIFT_RSTn
MCAN reset control .

enumerator kUTICK_RST_SHIFT_RSTn
UTICK reset control .

enumerator kFC0_RST_SHIFT_RSTn
FC0 reset control .

enumerator kFC1_RST_SHIFT_RSTn
FC1 reset control .

enumerator kFC2_RST_SHIFT_RSTn
FC2 reset control .

enumerator kFC3_RST_SHIFT_RSTn
FC3 reset control .

enumerator kFC4_RST_SHIFT_RSTn
FC4 reset control .

enumerator kFC5_RST_SHIFT_RSTn
FC5 reset control .

enumerator kFC6_RST_SHIFT_RSTn
FC6 reset control .

enumerator kFC7_RST_SHIFT_RSTn
FC7 reset control .

enumerator kDMIC_RST_SHIFT_RSTn
DMIC reset control .

enumerator kCTIMER2_RST_SHIFT_RSTn
TIMER2 reset control .

enumerator kUSB0_DEV_RST_SHIFT_RSTn
USB0_DEV reset control .

enumerator kCTIMER0_RST_SHIFT_RSTn
TIMER0 reset control .

enumerator kCTIMER1_RST_SHIFT_RSTn
TIMER1 reset control .

enumerator kDMA1_RST_SHIFT_RSTn
DMA1 reset control .

enumerator kCMP_RST_SHIFT_RSTn
CMP reset control .

enumerator kFREQME_RST_SHIFT_RSTn
FREQME reset control .

enumerator kCDOG_RST_SHIFT_RSTn
Code Watchdog reset control

enumerator kRNG_RST_SHIFT_RSTn
RNG reset control .

enumerator kSYSCTL_RST_SHIFT_RSTn
SYSCTL reset control .

enumerator kUSB0HMR_RST_SHIFT_RSTn
USB0HMR reset control

enumerator kUSB0HSL_RST_SHIFT_RSTn
USB0HSL reset control

enumerator kCSS_RST_SHIFT_RSTn
CSS reset control .

enumerator kPOWERQUAD_RST_SHIFT_RSTn
PowerQuad reset control .

enumerator kCTIMER3_RST_SHIFT_RSTn
TIMER3 reset control .

enumerator kCTIMER4_RST_SHIFT_RSTn
TIMER4 reset control .

enumerator kPUF_RST_SHIFT_RSTn
PUF reset control

enumerator kPKC_RST_SHIFT_RSTn
PKC reset control .

enumerator kANACTRL_RST_SHIFT_RSTn
ANACTRL reset control .

enumerator kHLSPI_RST_SHIFT_RSTn
HS LSPI reset control

enumerator kGPIOSEC_RST_SHIFT_RSTn
GPIO_SEC reset control .

enumerator kGPIO_SEC_INT_RST_SHIFT_RSTn
GPIO secure int reset control .

enumerator kI3C0_RST_SHIFT_RSTn
I3C0 reset control .

enumerator kENC0_RST_SHIFT_RSTn
ENC0 reset control .

enumerator kENC1_RST_SHIFT_RSTn
ENC1 reset control .

enumerator kPWM0_RST_SHIFT_RSTn
PWM0 reset control .

enumerator kPWM1_RST_SHIFT_RSTn
PWM1 reset control .

enumerator kAOI0_RST_SHIFT_RSTn
AOI0 reset control .

enumerator kAOI1_RST_SHIFT_RSTn
AOI1 reset control .

enumerator kFTM0_RST_SHIFT_RSTn
FTM0 reset control .

enumerator kDAC1_RST_SHIFT_RSTn
DAC1 reset control .

enumerator kDAC2_RST_SHIFT_RSTn
DAC2 reset control .

enumerator kOPAMP0_RST_SHIFT_RSTn
OPAMP0 reset control .

enumerator kOPAMP1_RST_SHIFT_RSTn
OPAMP1 reset control .

enumerator kOPAMP2_RST_SHIFT_RSTn
OPAMP2 reset control .

enumerator kHSCMP0_RST_SHIFT_RSTn
HSCMP0 reset control .

enumerator kHSCMP1_RST_SHIFT_RSTn
HSCMP1 reset control .

enumerator kHSCMP2_RST_SHIFT_RSTn
HSCMP2 reset control .

enumerator kVREF_RST_SHIFT_RSTn
VREF reset control .

typedef enum _SYSCON_RSTn SYSCON_RSTn_t
Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

typedef SYSCON_RSTn_t reset_ip_name_t

void RESET_SetPeripheralReset(*reset_ip_name_t* peripheral)

Assert reset to peripheral.

Asserts reset signal to specified peripheral module.

Parameters

- peripheral – Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_ClearPeripheralReset(*reset_ip_name_t* peripheral)

Clear reset to peripheral.

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

- peripheral – Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_PeripheralReset(*reset_ip_name_t* peripheral)

Reset peripheral module.

Reset peripheral module.

Parameters

- peripheral – Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void RESET_ReleasePeripheralReset(*reset_ip_name_t* peripheral)

Release peripheral module.

Release peripheral module.

Parameters

- peripheral – Peripheral to release. The enum argument contains encoding of reset register and reset bit position in the reset register.

FSL_RESET_DRIVER_VERSION
reset driver version 2.4.0

ADC_RSTS
Array initializers with peripheral reset bits

MCAN_RSTS

CRC_RSTS

CTIMER_RSTS

DMA_RSTS_N

FLEXCOMM_RSTS
GINT_RSTS
GPIO_RSTS_N
INPUTMUX_RSTS
IOCON_RSTS
FLASH_RSTS
MRT_RSTS
PINT_RSTS
CDOG_RSTS
RNG_RSTS
SCT_RSTS
USB0D_RST
USB0HMR_RST
USB0HSL_RST
UTICK_RSTS
WWDT_RSTS
OSTIMER_RSTS
I3C_RSTS
ENC_RSTS
PWM_RSTS
AOI_RSTS
DAC_RSTS
OPAMP_RSTS
HSCMP_RSTS
POWERQUAD_RSTS
PUF_RSTS
RTC_RSTS
USB0HSL_RSTS
USB0HMR_RSTS
USB0D_RSTS

2.76 RNG: Random Number Generator

FSL_RNG_DRIVER_VERSION

RNG driver version. Version 2.0.3.

Current version: 2.0.3

Change log:

- Version 2.0.0
 - Initial version
- Version 2.0.1
 - Fix MISRA C-2012 issue.
- Version 2.0.2
 - Add RESET_PeripheralReset function inside RNG_Init and RNG_Deinit functions.
- Version 2.0.3
 - Modified RNG_Init and RNG_GetRandomData functions, added rng_accumulateEntropy and rng_readEntropy functions.
 - These changes are reflecting recommended usage of RNG according to device UM.

void RNG_Init(RNG_Type *base)

Initializes the RNG.

This function initializes the RNG. When called, the RNG module and ring oscillator is enabled.

Parameters

- base – RNG base address

Returns

If successful, returns the kStatus_RNG_Success. Otherwise, it returns an error.

void RNG_Deinit(RNG_Type *base)

Shuts down the RNG.

This function shuts down the RNG.

Parameters

- base – RNG base address.

status_t RNG_GetRandomData(RNG_Type *base, void *data, size_t dataSize)

Gets random data.

This function gets random data from the RNG.

Parameters

- base – RNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

Returns

random data

static inline uint32_t RNG_GetRandomWord(RNG_Type *base)

Returns random 32-bit number.

This function gets random number from the RNG.

Parameters

- base – RNG base address.

Returns

random number

2.77 RTC: Real Time Clock

void RTC_WriteRegister (___IO uint32_t *reg, uint32_t value)

Write value to RTC register.

Note: This API will write data to RTC register, all bus synchornous operation will be done.

uint32_t RTC_ReadRegister (___I uint32_t *reg)

Read data from RTC register.

Note: This API will read the RTC's register value.

void RTC_Init(RTC_Type *base, rtc_config_t *config)

Inititalize the RTC mudule.

Note: This API should be called at the beginning of the application using the RTC driver. This API will help configure the calibration value, count down auto start, enable RTC timer and the initialized date for counting.

Parameters

- base – RTC peripheral base address
- config – configuration for initializing RTC.

static inline void RTC_Deinit(RTC_Type *base)

Stop the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

void RTC_GetDefaultConfig(RTC_Type *base, rtc_config_t *config)

Get default config for RTC setting. This configure structure will help setup calibration value, formated date value. And set if enable RTC timer and count down auto start function.

Parameters

- base – RTC peripheral base address
- config – Configure structure for setting RTC module. Refer to rtc_config_t.

status_t RTC_SetDatetime(RTC_Type *base, const rtc_datetime_t *datetime)

Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

- base – RTC peripheral base address

- `datetime` – Pointer to structure where the date and time details to set are stored

Returns

`kStatus_Success`: Success in setting the time and starting the RTC
`kStatus_InvalidArgument`: Error because the `datetime` format is incorrect

`void RTC_GetDatetime(RTC_Type *base, rtc_datetime_t *datetime)`

Gets the RTC time and stores it in the given time structure.

Parameters

- `base` – RTC peripheral base address
- `datetime` – Pointer to structure where the date and time details are stored.

`static inline void RTC_SetSleepTime(RTC_Type *base, uint32_t value)`

Sets the RTC sleep time.

The function will set the sleep time for counting down, if the value in SLEEPT register equals to 0, an interrupt will be generated if this interrupt is enabled.

Parameters

- `base` – RTC peripheral base address
- `value` – sleep time, uint is seconds.

`static inline uint32_t RTC_GetSleepTime(RTC_Type *base)`

Returns the current RTC sleep time value.

Parameters

- `base` – RTC peripheral base address

Returns

The sleep remaining in register.

`static inline void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)`

Enables the selected RTC interrupts.

Parameters

- `base` – RTC peripheral base address
- `mask` – The interrupts to enable. Refer to the enumeration `rtc_interrupt_enable_t`

`static inline void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)`

Disables the selected RTC interrupts.

Parameters

- `base` – RTC peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `rtc_interrupt_enable_t`

`static inline uint32_t RTC_GetStatusFlags(RTC_Type *base)`

Gets the RTC status flags.

Parameters

- `base` – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `rtc_status_flags_t`


```
static inline uint32_t RTC_GetRawInterruptStatus(RTC_Type *base)
```

Gets the RTC Raw interrupt status.

Parameters

- base – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `rtc_status_flags_t`

```
static inline uint32_t RTC_GetMaskedInterruptStatus(RTC_Type *base)
```

Gets the RTC status flags.

Parameters

- base – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `rtc_status_flags_t`

```
static inline void RTC_ClearMaskedInterruptStatus(RTC_Type *base, uint32_t mask)
```

Clear masked interrupt status.

Parameters

- base – RTC peripheral base address.
- mask – The interrupts to enable.

Returns

The status flags. This is the logical OR of members of the enumeration `rtc_masked_interrupt_status_t`

```
static inline void RTC_StartTimer(RTC_Type *base)
```

Starts the RTC time counter.

After calling this function, the timer counter increments once a second.

Parameters

- base – RTC peripheral base address

```
static inline void RTC_StopTimer(RTC_Type *base)
```

Stops the RTC time counter.

RTC's seconds register can be written to only when the timer is stopped.

Parameters

- base – RTC peripheral base address

```
void RTC_EnableSleepAutoStart(RTC_Type *base, bool enable)
```

Enable the sleep count down auto Start.

Parameters

- base – RTC peripheral base address
- enable – 1 for enable, 0 for disable.

```
static inline void RTC_StartSleep(RTC_Type *base)
```

Start the count down manually.

Parameters

- base – RTC peripheral base address

static inline void RTC_SetCalibrationValue(RTC_Type *base, uint32_t value)

Set calibration value. The calibration value decide the number of TFRO clock pulses in one tick.

Parameters

- base – RTC peripheral base address
- value – The value to be set.

static inline uint32_t RTC_GetCalibrationValue(RTC_Type *base)

Get the calibration value from register.

Parameters

- base – RTC peripheral base address

FSL_RTC_DRIVER_VERSION

Version 2.1.0

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_CountDownInterruptEnable

Count down interrupt.

enum _rtc_status_flags

List of RTC flags.

Values:

enumerator kRTC_EnabledFlag

Flag for RTC enabled or not.

enumerator kRTC_FreezeFlag

Flag for RTC freeze or not.

enumerator kRTC_RunFlag

Flag for RTC running status.

enum _rtc_raw_interrupt_status

List of RTC raw interrupt status.

Values:

enumerator kRTC_CountDownRawStatus

Raw interrupt status for count down finish.

enum _rtc_masked_interrupt_status

List of RTC masked interrupt status.

Values:

enumerator kRTC_CountDownMaskedStatus

Masked interrupt status for count down finish.

typedef enum _rtc_interrupt_enable rtc_interrupt_enable_t

List of RTC interrupts.

typedef enum _rtc_status_flags rtc_status_flags_t

List of RTC flags.

typedef enum _rtc_raw_interrupt_status rtc_raw_interrupt_status_t

List of RTC raw interrupt status.

```
typedef enum _rtc_masked_interrupt_status rtc_masked_interrupt_status_t
```

List of RTC masked interrupt status.

```
typedef struct _rtc_datetime rtc_datetime_t
```

Structure is used to hold the date and time.

```
typedef struct _rtc_config rtc_config_t
```

Structure used for configuration.

RTC_TFRO_FREQ

RTC_SYNC_TIME_OUT_TICKS

Time-out ticks is (in ARM clock ticks) that a bus synchronization wait should take. Worst-case wait is 3 TFRO ticks. Worst case ARM speed is full SFRO (no divider). Since the TFRO could be somewhat slower than spec'd, and the SFRO somewhat faster, we add 33% as margin - it is just for an emergency time-out.

```
struct _rtc_datetime
```

#include <fsl_rtc.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 _rtc_config
```

#include <fsl_rtc.h> Structure used for configuration.

Public Members

bool enableAutoStart

Enable count down auto start.

bool enableTimer

Enable RTC timer.

2.78 Runbootloader

```
void bootloader_user_entry(void *arg)
```

Run the Bootloader API to force into the ISP mode base on the user arg.

Parameters

- `arg` – Indicates API prototype fields definition. Refer to the above `user_app_boot_invoke_option_t` structure

FSL_ROMAPI_RUNBOOTLOADER_DRIVER_VERSION

ROMAPI_RUNBOOTLOADER driver version 2.0.0.

uint32_t reserved

uint32_t boot_image_index

uint32_t instance

uint32_t boot_interface

uint32_t mode

uint32_t tag

struct *user_app_boot_invoke_option_t* B

uint32_t U

union *user_app_boot_invoke_option_t* option

struct *user_app_boot_invoke_option_t*

#include <fsl_runbootloader.h>

union option

Public Members

struct *user_app_boot_invoke_option_t* B

uint32_t U

struct B

2.79 Sbloader

enum _sectionType

sb3 section definitions

section type

Values:

enumerator kSectionNone

end or invalid

enumerator kSectionDataRange

enumerator kSectionDiffUpdate

enumerator kSectionDDRConfig

enumerator kSectionRegister

Values:

enumerator kFwVerChk_Id_none

enumerator kFwVerChk_Id_nonsecure

enumerator kFwVerChk_Id_secure

enum _loader_command_sb3

loader command enum

Values:

enumerator kSB3_CmdInvalid

enumerator kSB3_CmdErase

enumerator kSB3_CmdLoad

enumerator kSB3_CmdExecute

enumerator kSB3_CmdCall

enumerator kSB3_CmdProgramFuse

enumerator kSB3_CmdProgramIFR

enumerator kSB3_CmdLoadCmac

enumerator kSB3_CmdCopy

enumerator kSB3_CmdLoadHashLocking

enumerator kSB3_CmdLoadKeyBlob

enumerator kSB3_CmdConfigMem

enumerator kSB3_CmdFillMem

enumerator kSB3_CmdFwVerCheck

enum kb_operation_t

Details of the operation to be performed by the ROM.

The kRomAuthenticateImage operation requires the entire signed image to be available to the application.

Values:

enumerator kRomAuthenticateImage

Authenticate a signed image.

enumerator kRomLoadImage

Load SB file.

enumerator kRomOperationCount

SB loader status codes.

Values:

enumerator kStatusRomLdrSectionOverrun

enumerator kStatusRomLdrSignature

enumerator kStatusRomLdrSectionLength

enumerator kStatusRomLdrUnencryptedOnly

```
enumerator kStatusRomLdrEOFReached
enumerator kStatusRomLdrChecksum
enumerator kStatusRomLdrCrc32Error
enumerator kStatusRomLdrUnknownCommand
enumerator kStatusRomLdrIdNotFound
enumerator kStatusRomLdrDataUnderrun
enumerator kStatusRomLdrJumpReturned
enumerator kStatusRomLdrCallFailed
enumerator kStatusRomLdrKeyNotFound
enumerator kStatusRomLdrSecureOnly
enumerator kStatusRomLdrResetReturned
enumerator kStatusRomLdrRollbackBlocked
enumerator kStatusRomLdrInvalidSectionMacCount
enumerator kStatusRomLdrUnexpectedCommand
enumerator kStatusRomLdrBadSBKEK
enumerator kStatusRomLdrPendingJumpCommand

enum _fw_version_check_option
    Values:
        enumerator kRomLdr_FwCheckOption_SecureVersion
        enumerator kRomLdr_FwCheckOption_NonSecureVersion

typedef uint8_t chunk_v3_t[16]

typedef struct _ldr_buf ldr_buf_t

typedef struct _ldr_Context_v3 ldr_Context_v3_t
    Provides forward reference to the loader context definition.

typedef status_t (*pLdrFnc_v3_t)(ldr_Context_v3_t *content)
    Function pointer definition for all loader action functions.

typedef enum _sectionType section_type_t
    sb3 section definitions
    section type

typedef struct range_header sb3_data_range_header_t
    section data range structure

typedef struct range_header_expansion sb3_data_range_expansion_t

typedef struct copy_memory_expansion sb3_copy_memory_expansion_t

typedef struct copy sb3_copy_memory_t

typedef struct load_keyblob sb3_load_keyblob_t
```

```
typedef struct fill_memory_expansion sb3_fill_memory_expansion_t
typedef struct fill_memory sb3_fill_memory_t
typedef struct config_memory sb3_config_memory_t
typedef struct fw_ver_check sb3_fw_ver_check_t
typedef struct section_header sb3_section_header_t
    sb3 DATA section header format
typedef enum _loader_command_sb3 sb3_cmd_t
    loader command enum
typedef uint8_t chunk_t[16]
typedef struct _boot_cmd boot_cmd_t
    Boot command definition.
typedef struct _boot_hdr1 boot_hdr1_t
    Definition for boot image file header chunk 1.
typedef struct _boot_hdr2 boot_hdr2_t
    Definition for boot image file header chunk 2.
typedef struct _ldr_Context ldr_Context_t
    Provides forward reference to the loader context definition.
typedef status_t (*pLdrFnc_t)(ldr_Context_t *context)
    Function pointer definition for all loader action functions.
typedef status_t (*pJumpFnc_t)(uint32_t parameter)
    Jump command function pointer definition.
typedef status_t (*pCallFnc_t)(uint32_t parameter, uint32_t *func)
    Call command function pointer definition.
typedef struct Crc32Data crc32_data_t
    State information for the CRC32 algorithm.
typedef struct soc_memory_map_struct soc_mem_regions_t
typedef struct arena_context arena_context_t
typedef struct mem_region mem_region_t
    Memory region information table.
typedef struct memory_attribute_struct mem_attribute_t
    Memory Attribute Structure.
typedef struct memory_context_struct mem_context_t
    Memory context structure.
typedef struct api_memory_region_interface api_memory_region_interface_t
    Memory region interface structure.
typedef struct memory_map_entry api_memory_map_entry_t
    Memory entry data structure.
typedef struct api_core_context api_core_context_t
    The API context structure.
```

status_t Sbloder_Init(*api_core_context_t* *ctx)

Perform the Sbloder runtime environment initialization This API is used for initializing the sbloder state machine before calling the *api_sbloder_pump*. This API should be called after the *iap_api_init* API.

Parameters

- ctx – Pointer to IAP API core context structure.

Return values

kStatus_Success – Api was executed succesfully.

status_t Sbloder_Pump(*api_core_context_t* *ctx, *uint8_t* *data, *uint32_t* length)

Handle the SB data stream This API is used for handling the secure binary(SB3.1 format) data stream, which is used for image update, lifecycle advancing, etc. This API should be called after the *iap_api_init* and *api_sbloder_init* APIs.

Parameters

- ctx – Pointer to IAP API core context structure.
- data – Pointer to source data that is the sb file buffer data.
- length – The size of the process buffer data.

Return values

- kStatus_Success – Api was executed succesfully.
- kStatus_InvalidArgument – An invalid argument is provided.
- kStatus_Fail – API execution failed.

status_t Sbloder_Finalize(*api_core_context_t* *ctx)

Finish the sbloder handling The API is used for finalizing the sbloder operations.

Parameters

- ctx – Pointer to IAP API core context structure.

Return values

kStatus_Success – Api was executed succesfully.

SB3_BYTES_PER_CHUNK

Defines the number of bytes in a cipher block (chunk). This is dictated by the encryption algorithm.

SB3_DATA_RANGE_HEADER_FLAGS_ERASE_MASK

bit 0

SB3_DATA_RANGE_HEADER_FLAGS_LOAD_MASK

bit 1

SBLOADER_V3_CMD_SET_ALL

The all of the allowed command.

SBLOADER_V3_CMD_SET_IN_ISP_MODE

The allowed command set in ISP mode.

SBLOADER_V3_CMD_SET_IN_REC_MODE

The allowed command set in recovery mode.

SB3_DATA_BUFFER_SIZE_IN_BYTE

SB_FILE_MAJOR_VERSION

Determines the version of SB loader implementation (1: sb1.0; 2: sb2.0; 3.1: sb3.1)

SB_FILE_MINOR_VERSION

kStatusGroup_SBLoader

Bootloader status group numbers.

RAM_REGION_COUNT

Contiguous RAM region count.

FLASH_REGION_COUNT

Contiguous FLASH region count.

FFR_REGION_COUNT

Contiguous FFR region count.

MEM_INTERFACE_COUNT

Memory Interface count.

FLEXSPINOR_REGION_COUNT

Contiguous FLEXSPINOR meomry count.

BYTES_PER_CHUNK

Defines the number of bytes in a cipher block (chunk). This is dictated by the encryption algorithm.

SB_SECTION_COUNT_MAX

BOOT_SIGNATURE

Boot image signature in 32-bit little-endian format "PMTS".

BOOT_SIGNATURE2

Boot image signature in 32-bit little-endian format "ltgs".

FFLG_DISPLAY_PROGRESS

These define file header flags.

SFLG_SECTION_BOOTABLE

These define section header flags.

CFLG_LAST_TAG

These define boot command flags.

ROM_ERASE_ALL_MASK

ROM_ERASE_CMD flags.

ROM_ERASE_ALL_UNSECURE_MASK

ROM_JUMP_SP_MASK

ROM_JUMP_CMD flags.

ROM_MEM_DEVICE_ID_SHIFT

Memory device id shift at sb command flags.

ROM_MEM_DEVICE_ID_MASK

Memory device id mask.

ROM_MEM_GROUP_ID_SHIFT

Memory group id shift at sb command flags.

ROM_MEM_GROUP_ID_MASK

Memory group id flags mask.

ROM_PROG_8BYTE_MASK

ROM_PROG_CMD flags.

ROM_NOP_CMD

These define the boot command tags.

ROM_TAG_CMD

ROM_LOAD_CMD

ROM_FILL_CMD

ROM_JUMP_CMD

ROM_CALL_CMD

ROM_MODE_CMD

ROM_ERASE_CMD

ROM_RESET_CMD

ROM_MEM_ENABLE_CMD

ROM_PROG_CMD

ROM_FW_VER_CHK

SBLOADER_CMD_SET_IN_ISP_MODE

SBLOADER_CMD_SET_IN_REC_MODE

ROM_BOOT_SECTION_ID

Plugin return codes.

ROM_BOOT_IMAGE_ID

struct _ldr_buf

#include <fsl_sbloader_v3.h>

struct range_header

#include <fsl_sbloader_v3.h> section data range structure

struct range_header_expansion

#include <fsl_sbloader_v3.h>

struct copy_memory_expansion

#include <fsl_sbloader_v3.h>

struct copy

#include <fsl_sbloader_v3.h>

struct load_keyblob

#include <fsl_sbloader_v3.h>

struct fill_memory_expansion

#include <fsl_sbloader_v3.h>

Public Members

uint32_t pattern

word to be used as pattern

struct fill_memory

#include <fsl_sbloader_v3.h>

struct config_memory

#include <fsl_sbloader_v3.h>

Public Members

`uint32_t` `address`
address of config blob

`struct fw__ver__check`
`#include <fsl_sbloader_v3.h>`

`struct section_header`
`#include <fsl_sbloader_v3.h>` sb3 DATA section header format

`struct kb_region_t`
`#include <fsl_sbloader_v3.h>` Memory region definition.

`struct kb_load_sb_t`
`#include <fsl_sbloader_v3.h>`

`struct kb_authenticate_t`
`#include <fsl_sbloader_v3.h>`

`struct kb_options_t`
`#include <fsl_sbloader_v3.h>`

Public Members

`uint32_t` `version`
Should be set to KbootApiVersion.

`uint8_t *buffer`
Caller-provided buffer used by Kboot.

`struct __ldr_Context_v3`
`#include <fsl_sbloader_v3.h>` Loader context definition.

Public Members

`pLdrFnc_v3_t` `Action`
pointer to loader action function

`uint32_t` `block_size`
size of each block in bytes

`uint32_t` `block_data_size`
data size in bytes (NBOOT_SB3_CHUNK_SIZE_IN_BYTES)

`uint32_t` `block_data_total`
data max size in bytes (`block_size * data_size`)

`uint32_t` `block_buffer_size`
block0 and block size

`uint32_t` `processedBlocks`
will be used for both block0 and blockx

`bool` `in_data_block`
data block offset in a block. in progress of handling a data block within a block

`bool` `in_data_section`
in progress of handling a data section within a data block

`bool in_data_range`
in progress of handling a data range within a data section

`uint32_t commandSet`
support command set during sb file handling

`uint8_t data_buffer[(MAX(128, NBOOT_KEY_BLOB_SIZE_IN_BYTE_MAX))]`
temporary data buffer

`kb_options_t fromAPI`
options from ROM API

`struct __boot_cmd`
#include <fsl_sbloader.h> Boot command definition.

Public Members

`uint8_t checksum`
8-bit checksum over command chunk

`uint8_t tag`
command tag (identifier)

`uint16_t flags`
command flags (modifier)

`uint32_t address`
address argument

`uint32_t count`
count argument

`uint32_t data`
data argument

`struct __boot_hdr1`
#include <fsl_sbloader.h> Definition for boot image file header chunk 1.

Public Members

`uint32_t hash`
last 32-bits of SHA-1 hash

`uint32_t signature`
must equal “STMP”

`uint8_t major`
major file format version

`uint8_t minor`
minor file format version

`uint16_t fileFlags`
global file flags

`uint32_t fileChunks`
total chunks in the file

`struct __boot_hdr2`
#include <fsl_sbloader.h> Definition for boot image file header chunk 2.

Public Members

uint32_t bootOffset
 chunk offset to the first boot section

uint32_t bootSectID
 section ID of the first boot section

uint16_t keyCount
 number of keys in the key dictionary

uint16_t keyOffset
 chunk offset to the key dictionary

uint16_t hdrChunks
 number of chunks in the header

uint16_t sectCount
 number of sections in the image

struct Crc32Data

#include <fsl_sbloader.h> State information for the CRC32 algorithm.

Public Members

uint32_t currentCrc
 Current CRC value.

uint32_t byteCountCrc
 Number of bytes processed.

struct _ldr_Context

#include <fsl_sbloader.h> Loader context definition.

Public Members

pLdrFnc_t Action
 pointer to loader action function

uint32_t fileChunks
 chunks remaining in file

uint32_t sectChunks
 chunks remaining in section

uint32_t bootSectChunks
 number of chunks we need to complete the boot section

uint32_t receivedChunks
 number of chunks we need to complete the boot section

uint16_t fileFlags
 file header flags

uint16_t keyCount
 number of keys in the key dictionary

uint32_t objectID
 ID of the current boot section or image

crc32_data_t crc32
crc calculated over load command payload

uint8_t *src
source buffer address

chunk_t initVector
decryption initialization vector

chunk_t dek
chunk size DEK if the image is encrypted

chunk_t scratchPad
chunk size scratch pad area

boot_cmd_t bootCmd
current boot command

uint32_t skipCount
Number of chunks to skip

bool skipToEnd
true if skipping to end of file

uint32_t offsetSignatureBytes
offset to signagure block header in bytesn

struct soc__memory__map__struct
#include <fsl_sbloader.h>

struct arena__context
#include <fsl_sbloader.h>

struct mem__region
#include <fsl_sbloader.h> Memory region information table.

struct memory__attribute__struct
#include <fsl_sbloader.h> Memory Attribute Structure.

struct memory__context__struct
#include <fsl_sbloader.h> Memory context structure.

struct api__memory__region__interface
#include <fsl_sbloader.h> Memory region interface structure.

struct memory__map__entry
#include <fsl_sbloader.h> Memory entry data structure.

struct api__core__context
#include <fsl_sbloader.h> The API context structure.

union __unnamed36__

Public Members

kb_authenticate_t authenticate
Settings for kb_authenticate_t operation.

kb_load_sb_t loadSB
Settings for kb_load_sb_t operation.

struct ramRegions

struct flashRegions

struct ffrRegions

struct flexspiNorRegions

2.80 SCTimer: SCTimer/PWM (SCT)

status_t SCTIMER_Init(SCT_Type *base, const *sctimer_config_t* *config)

Un gates the SCTimer clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SCTimer driver.

Parameters

- base – SCTimer peripheral base address
- config – Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

void SCTIMER_Deinit(SCT_Type *base)

Gates the SCTimer clock.

Parameters

- base – SCTimer peripheral base address

void SCTIMER_GetDefaultConfig(*sctimer_config_t* *config)

Fills in the SCTimer configuration structure with the default settings.

The default values are:

```
config->enableCounterUnify = true;
config->clockMode = kSCTIMER_System_ClockMode;
config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
config->enableBidirection_l = false;
config->enableBidirection_h = false;
config->prescale_l = 0U;
config->prescale_h = 0U;
config->outInitState = 0U;
config->inputsync = 0xFU;
```

Parameters

- config – Pointer to the user configuration structure.

status_t SCTIMER_SetupPwm(SCT_Type *base, const *sctimer_pwm_signal_param_t* *pwmParams, *sctimer_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event)

Configures the PWM signal parameters.

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events

are enabled for the same state. The state number can be retrieved by calling the function `SCTIMER_GetCurrentStateNumber()`. The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

Note: When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's `pwmFreq_Hz`.

Parameters

- `base` – SCTimer peripheral base address
- `pwmParams` – PWM parameters to configure the output
- `mode` – PWM operation mode, options available in enumeration `sctimer_pwm_mode_t`
- `pwmFreq_Hz` – PWM signal frequency in Hz
- `srcClock_Hz` – SCTimer counter clock in Hz
- `event` – Pointer to a variable where the PWM period event number is stored

Returns

`kStatus_Success` on success `kStatus_Fail` If we have hit the limit in terms of number of events created or if an incorrect PWM duty cycle is passed in.

```
void SCTIMER_UpdatePwmDutycycle(SCT_Type *base, sctimer_out_t output, uint8_t  
                                dutyCyclePercent, uint32_t event)
```

Updates the duty cycle of an active PWM signal.

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

Parameters

- `base` – SCTimer peripheral base address
- `output` – The output to configure
- `dutyCyclePercent` – New PWM pulse width; the value should be between 1 to 100
- `event` – Event number associated with this PWM signal. This was returned to the user by the function `SCTIMER_SetupPwm()`.

```
static inline void SCTIMER_EnableInterrupts(SCT_Type *base, uint32_t mask)
```

Enables the selected SCTimer interrupts.

Parameters

- `base` – SCTimer peripheral base address
- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `sctimer_interrupt_enable_t`

```
static inline void SCTIMER_DisableInterrupts(SCT_Type *base, uint32_t mask)
```

Disables the selected SCTimer interrupts.

Parameters

- `base` – SCTimer peripheral base address

- `mask` – The interrupts to enable. This is a logical OR of members of the enumeration `sctimer_interrupt_enable_t`

`static inline uint32_t SCTIMER_GetEnabledInterrupts(SCT_Type *base)`

Gets the enabled SCTimer interrupts.

Parameters

- `base` – SCTimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `sctimer_interrupt_enable_t`

`static inline uint32_t SCTIMER_GetStatusFlags(SCT_Type *base)`

Gets the SCTimer status flags.

Parameters

- `base` – SCTimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `sctimer_status_flags_t`

`static inline void SCTIMER_ClearStatusFlags(SCT_Type *base, uint32_t mask)`

Clears the SCTimer status flags.

Parameters

- `base` – SCTimer peripheral base address
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `sctimer_status_flags_t`

`static inline void SCTIMER_StartTimer(SCT_Type *base, uint32_t countertoStart)`

Starts the SCTimer counter.

Note: In 16-bit mode, we can enable both Counter_L and Counter_H, In 32-bit mode, we only can select Counter_U.

Parameters

- `base` – SCTimer peripheral base address
- `countertoStart` – The SCTimer counters to enable. This is a logical OR of members of the enumeration `sctimer_counter_t`.

`static inline void SCTIMER_StopTimer(SCT_Type *base, uint32_t countertoStop)`

Halts the SCTimer counter.

Parameters

- `base` – SCTimer peripheral base address
- `countertoStop` – The SCTimer counters to stop. This is a logical OR of members of the enumeration `sctimer_counter_t`.

`status_t SCTIMER_CreateAndScheduleEvent(SCT_Type *base, sctimer_event_t howToMonitor, uint32_t matchValue, uint32_t whichIO, sctimer_counter_t whichCounter, uint32_t *event)`

Create an event that is triggered on a match or IO and schedule in current state.

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The

event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

- `base` – SCTimer peripheral base address
- `howToMonitor` – Event type; options are available in the enumeration `sctimer_interrupt_enable_t`
- `matchValue` – The match value that will be programmed to a match register
- `whichIO` – The input or output that will be involved in event triggering. This field is ignored if the event type is “match only”
- `whichCounter` – SCTimer counter to use. In 16-bit mode, we can select `Counter_L` and `Counter_H`, In 32-bit mode, we can select `Counter_U`.
- `event` – Pointer to a variable where the new event number is stored

Returns

`kStatus_Success` on success `kStatus_Error` if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

`void SCTIMER_ScheduleEvent(SCT_Type *base, uint32_t event)`

Enable an event in the current state.

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function `SCTIMER_SetupPwm()` or function `SCTIMER_CreateAndScheduleEvent()`.

Parameters

- `base` – SCTimer peripheral base address
- `event` – Event number to enable in the current state

`status_t SCTIMER_IncreaseState(SCT_Type *base)`

Increase the state by 1.

All future events created by calling the function `SCTIMER_ScheduleEvent()` will be enabled in this new state.

Parameters

- `base` – SCTimer peripheral base address

Returns

`kStatus_Success` on success `kStatus_Error` if we have hit the limit in terms of states used

`uint32_t SCTIMER_GetCurrentState(SCT_Type *base)`

Provides the current state.

User can use this to set the next state by calling the function `SCTIMER_SetupNextStateAction()`.

Parameters

- `base` – SCTimer peripheral base address

Returns

The current state

`static inline void SCTIMER_SetCounterState(SCT_Type *base, sctimer_counter_t whichCounter, uint32_t state)`

Set the counter current state.

The function is to set the state variable bit field of STATE register. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- state – The counter current state number (only support range from 0~31).

```
static inline uint16_t SCTIMER_GetCounterState(SCT_Type *base, sctimer_counter_t  
                                              whichCounter)
```

Get the counter current state value.

The function is to get the state variable bit field of STATE register.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The the counter current state value.

```
status_t SCTIMER_SetupCaptureAction(SCT_Type *base, sctimer_counter_t whichCounter,  
                                    uint32_t *captureRegister, uint32_t event)
```

Setup capture of the counter value on trigger of a selected event.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- captureRegister – Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
- event – Event number that will trigger the capture

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of match/capture registers available

```
void SCTIMER_SetCallback(SCT_Type *base, sctimer_event_callback_t callback, uint32_t event)
```

Receive notification when the event trigger an interrupt.

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

- base – SCTimer peripheral base address
- event – Event number that will trigger the interrupt
- callback – Function to invoke when the event is triggered

```
static inline void SCTIMER_SetupStateLdMethodAction(SCT_Type *base, uint32_t event, bool
                                                    fgLoad)
```

Change the load method of transition to the specified state.

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- event – Event number that will change the method to trigger the state transition
- fgLoad – The method to load highest-numbered event occurring for that state to the STATE register.
 - true: Load the STATEV value to STATE when the event occurs to be the next state.
 - false: Add the STATEV value to STATE when the event occurs to be the next state.

```
static inline void SCTIMER_SetupNextStateActionwithLdMethod(SCT_Type *base, uint32_t
                                                            nextState, uint32_t event, bool
                                                            fgLoad)
```

Transition to the specified state with Load method.

This transition will be triggered by the event number that is passed in by the user, the method decide how to load the highest-numbered event occurring for that state to the STATE register.

Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition
- fgLoad – The method to load the highest-numbered event occurring for that state to the STATE register.
 - true: Load the STATEV value to STATE when the event occurs to be the next state.
 - false: Add the STATEV value to STATE when the event occurs to be the next state.

```
static inline void SCTIMER_SetupNextStateAction(SCT_Type *base, uint32_t nextState, uint32_t
                                                event)
```

Transition to the specified state.

Deprecated:

Do not use this function. It has been superceded by SCTIMER_SetupNextStateActionwithLdMethod

This transition will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition

```
static inline void SCTIMER_SetupEventActiveDirection(SCT_Type *base,  
                                                    sctimer_event_active_direction_t  
                                                    activeDirection, uint32_t event)
```

Setup event active direction when the counters are operating in BIDIR mode.

Parameters

- base – SCTimer peripheral base address
- activeDirection – Event generation active direction, see *sctimer_event_active_direction_t*.
- event – Event number that need setup the active direction.

```
static inline void SCTIMER_SetupOutputSetAction(SCT_Type *base, uint32_t whichIO, uint32_t  
                                              event)
```

Set the Output.

This output will be set when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to set
- event – Event number that will trigger the output change

```
static inline void SCTIMER_SetupOutputClearAction(SCT_Type *base, uint32_t whichIO,  
                                                uint32_t event)
```

Clear the Output.

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to clear
- event – Event number that will trigger the output change

```
void SCTIMER_SetupOutputToggleAction(SCT_Type *base, uint32_t whichIO, uint32_t event)
```

Toggle the output level.

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to toggle
- event – Event number that will trigger the output change

```
static inline void SCTIMER_SetupCounterLimitAction(SCT_Type *base, sctimer_counter_t  
                                                  whichCounter, uint32_t event)
```

Limit the running counter.

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be limited

```
static inline void SCTIMER_SetupCounterStopAction(SCT_Type *base, sctimer_counter_t
                                                    whichCounter, uint32_t event)
```

Stop the running counter.

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be stopped

```
static inline void SCTIMER_SetupCounterStartAction(SCT_Type *base, sctimer_counter_t
                                                    whichCounter, uint32_t event)
```

Re-start the stopped counter.

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to re-start

```
static inline void SCTIMER_SetupCounterHaltAction(SCT_Type *base, sctimer_counter_t
                                                    whichCounter, uint32_t event)
```

Halt the running counter.

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the SCTIMER_StartTimer() function.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be halted

```
static inline void SCTIMER_SetupDmaTriggerAction(SCT_Type *base, uint32_t dmaNumber,
                                                  uint32_t event)
```

Generate a DMA request.

DMA request will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- dmaNumber – The DMA request to generate
- event – Event number that will trigger the DMA request

```
static inline void SCTIMER_SetCOUNTValue(SCT_Type *base, sctimer_counter_t whichCounter,
                                           uint32_t value)
```

Set the value of counter.

The function is to set the value of Count register, Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- `base` – SCTimer peripheral base address
- `whichCounter` – SCTimer counter to use. In 16-bit mode, we can select `Counter_L` and `Counter_H`, In 32-bit mode, we can select `Counter_U`.
- `value` – the counter value update to the `COUNT` register.

```
static inline uint32_t SCTIMER_GetCOUNTValue(SCT_Type *base, sctimer_counter_t  
                                              whichCounter)
```

Get the value of counter.

The function is to read the value of Count register, software can read the counter registers at any time..

Parameters

- `base` – SCTimer peripheral base address
- `whichCounter` – SCTimer counter to use. In 16-bit mode, we can select `Counter_L` and `Counter_H`, In 32-bit mode, we can select `Counter_U`.

Returns

The value of counter selected.

```
static inline void SCTIMER_SetEventInState(SCT_Type *base, uint32_t event, uint32_t state)
```

Set the state mask bit field of `EV_STATE` register.

Parameters

- `base` – SCTimer peripheral base address
- `event` – The `EV_STATE` register be set.
- `state` – The state value in which the event is enabled to occur.

```
static inline void SCTIMER_ClearEventInState(SCT_Type *base, uint32_t event, uint32_t state)
```

Clear the state mask bit field of `EV_STATE` register.

Parameters

- `base` – SCTimer peripheral base address
- `event` – The `EV_STATE` register be clear.
- `state` – The state value in which the event is disabled to occur.

```
static inline bool SCTIMER_GetEventInState(SCT_Type *base, uint32_t event, uint32_t state)
```

Get the state mask bit field of `EV_STATE` register.

Note: This function is to check whether the event is enabled in a specific state.

Parameters

- `base` – SCTimer peripheral base address
- `event` – The `EV_STATE` register be read.
- `state` – The state value.

Returns

The the state mask bit field of `EV_STATE` register.

- `true`: The event is enable in state.
- `false`: The event is disable in state.

```
static inline uint32_t SCTIMER_GetCaptureValue(SCT_Type *base, sctimer_counter_t  
                                              whichCounter, uint8_t capChannel)
```

Get the value of capture register.

This function returns the captured value upon occurrence of the events selected by the corresponding Capture Control registers occurred.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- capChannel – SCTimer capture register of capture channel.

Returns

The SCTimer counter value at which this register was last captured.

```
void SCTIMER_EventHandleIRQ(SCT_Type *base)
```

SCTimer interrupt handler.

Parameters

- base – SCTimer peripheral base address.

```
FSL_SCTIMER_DRIVER_VERSION
```

Version

```
enum __sctimer_pwm_mode
```

SCTimer PWM operation modes.

Values:

```
enumerator kSCTIMER_EdgeAlignedPwm
```

Edge-aligned PWM

```
enumerator kSCTIMER_CenterAlignedPwm
```

Center-aligned PWM

```
enum __sctimer_counter
```

SCTimer counters type.

Values:

```
enumerator kSCTIMER_Counter_L
```

16-bit Low counter.

```
enumerator kSCTIMER_Counter_H
```

16-bit High counter.

```
enumerator kSCTIMER_Counter_U
```

32-bit Unified counter.

```
enum __sctimer_input
```

List of SCTimer input pins.

Values:

```
enumerator kSCTIMER_Input_0
```

SCTIMER input 0

```
enumerator kSCTIMER_Input_1
```

SCTIMER input 1

enumerator kSCTIMER_Input_2

SCTIMER input 2

enumerator kSCTIMER_Input_3

SCTIMER input 3

enumerator kSCTIMER_Input_4

SCTIMER input 4

enumerator kSCTIMER_Input_5

SCTIMER input 5

enumerator kSCTIMER_Input_6

SCTIMER input 6

enumerator kSCTIMER_Input_7

SCTIMER input 7

enum _sctimer_out

List of SCTimer output pins.

Values:

enumerator kSCTIMER_Out_0

SCTIMER output 0

enumerator kSCTIMER_Out_1

SCTIMER output 1

enumerator kSCTIMER_Out_2

SCTIMER output 2

enumerator kSCTIMER_Out_3

SCTIMER output 3

enumerator kSCTIMER_Out_4

SCTIMER output 4

enumerator kSCTIMER_Out_5

SCTIMER output 5

enumerator kSCTIMER_Out_6

SCTIMER output 6

enumerator kSCTIMER_Out_7

SCTIMER output 7

enumerator kSCTIMER_Out_8

SCTIMER output 8

enumerator kSCTIMER_Out_9

SCTIMER output 9

enum _sctimer_pwm_level_select

SCTimer PWM output pulse mode: high-true, low-true or no output.

Values:

enumerator kSCTIMER_LowTrue

Low true pulses

enumerator kSCTIMER_HighTrue

High true pulses

enum `_sctimer_clock_mode`

SCTimer clock mode options.

Values:

enumerator `kSCTIMER_System_ClockMode`

System Clock Mode

enumerator `kSCTIMER_Sampled_ClockMode`

Sampled System Clock Mode

enumerator `kSCTIMER_Input_ClockMode`

SCT Input Clock Mode

enumerator `kSCTIMER_Asynchronous_ClockMode`

Asynchronous Mode

enum `_sctimer_clock_select`

SCTimer clock select options.

Values:

enumerator `kSCTIMER_Clock_On_Rise_Input_0`

Rising edges on input 0

enumerator `kSCTIMER_Clock_On_Fall_Input_0`

Falling edges on input 0

enumerator `kSCTIMER_Clock_On_Rise_Input_1`

Rising edges on input 1

enumerator `kSCTIMER_Clock_On_Fall_Input_1`

Falling edges on input 1

enumerator `kSCTIMER_Clock_On_Rise_Input_2`

Rising edges on input 2

enumerator `kSCTIMER_Clock_On_Fall_Input_2`

Falling edges on input 2

enumerator `kSCTIMER_Clock_On_Rise_Input_3`

Rising edges on input 3

enumerator `kSCTIMER_Clock_On_Fall_Input_3`

Falling edges on input 3

enumerator `kSCTIMER_Clock_On_Rise_Input_4`

Rising edges on input 4

enumerator `kSCTIMER_Clock_On_Fall_Input_4`

Falling edges on input 4

enumerator `kSCTIMER_Clock_On_Rise_Input_5`

Rising edges on input 5

enumerator `kSCTIMER_Clock_On_Fall_Input_5`

Falling edges on input 5

enumerator `kSCTIMER_Clock_On_Rise_Input_6`

Rising edges on input 6

enumerator `kSCTIMER_Clock_On_Fall_Input_6`

Falling edges on input 6

enumerator kSCTIMER_Clock_On_Rise_Input_7

Rising edges on input 7

enumerator kSCTIMER_Clock_On_Fall_Input_7

Falling edges on input 7

enum _sctimer_conflict_resolution

SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Values:

enumerator kSCTIMER_ResolveNone

No change

enumerator kSCTIMER_ResolveSet

Set output

enumerator kSCTIMER_ResolveClear

Clear output

enumerator kSCTIMER_ResolveToggle

Toggle output

enum _sctimer_event_active_direction

List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

Values:

enumerator kSCTIMER_ActiveIndependent

This event is triggered regardless of the count direction.

enumerator kSCTIMER_ActiveInCountUp

This event is triggered only during up-counting when BIDIR = 1.

enumerator kSCTIMER_ActiveInCountDown

This event is triggered only during down-counting when BIDIR = 1.

enum _sctimer_event

List of SCTimer event types.

Values:

enumerator kSCTIMER_InputLowOrMatchEvent

enumerator kSCTIMER_InputRiseOrMatchEvent

enumerator kSCTIMER_InputFallOrMatchEvent

enumerator kSCTIMER_InputHighOrMatchEvent

enumerator kSCTIMER_MatchEventOnly

enumerator kSCTIMER_InputLowEvent

enumerator kSCTIMER_InputRiseEvent

enumerator kSCTIMER_InputFallEvent

enumerator kSCTIMER_InputHighEvent

enumerator kSCTIMER_InputLowAndMatchEvent

enumerator kSCTIMER_InputRiseAndMatchEvent
enumerator kSCTIMER_InputFallAndMatchEvent
enumerator kSCTIMER_InputHighAndMatchEvent
enumerator kSCTIMER_OutputLowOrMatchEvent
enumerator kSCTIMER_OutputRiseOrMatchEvent
enumerator kSCTIMER_OutputFallOrMatchEvent
enumerator kSCTIMER_OutputHighOrMatchEvent
enumerator kSCTIMER_OutputLowEvent
enumerator kSCTIMER_OutputRiseEvent
enumerator kSCTIMER_OutputFallEvent
enumerator kSCTIMER_OutputHighEvent
enumerator kSCTIMER_OutputLowAndMatchEvent
enumerator kSCTIMER_OutputRiseAndMatchEvent
enumerator kSCTIMER_OutputFallAndMatchEvent
enumerator kSCTIMER_OutputHighAndMatchEvent

enum _sctimer_interrupt_enable

List of SCTimer interrupts.

Values:

enumerator kSCTIMER_Event0InterruptEnable
 Event 0 interrupt
enumerator kSCTIMER_Event1InterruptEnable
 Event 1 interrupt
enumerator kSCTIMER_Event2InterruptEnable
 Event 2 interrupt
enumerator kSCTIMER_Event3InterruptEnable
 Event 3 interrupt
enumerator kSCTIMER_Event4InterruptEnable
 Event 4 interrupt
enumerator kSCTIMER_Event5InterruptEnable
 Event 5 interrupt
enumerator kSCTIMER_Event6InterruptEnable
 Event 6 interrupt
enumerator kSCTIMER_Event7InterruptEnable
 Event 7 interrupt
enumerator kSCTIMER_Event8InterruptEnable
 Event 8 interrupt
enumerator kSCTIMER_Event9InterruptEnable
 Event 9 interrupt

enumerator kSCTIMER_Event10InterruptEnable

Event 10 interrupt

enumerator kSCTIMER_Event11InterruptEnable

Event 11 interrupt

enumerator kSCTIMER_Event12InterruptEnable

Event 12 interrupt

enum _sctimer_status_flags

List of SCTimer flags.

Values:

enumerator kSCTIMER_Event0Flag

Event 0 Flag

enumerator kSCTIMER_Event1Flag

Event 1 Flag

enumerator kSCTIMER_Event2Flag

Event 2 Flag

enumerator kSCTIMER_Event3Flag

Event 3 Flag

enumerator kSCTIMER_Event4Flag

Event 4 Flag

enumerator kSCTIMER_Event5Flag

Event 5 Flag

enumerator kSCTIMER_Event6Flag

Event 6 Flag

enumerator kSCTIMER_Event7Flag

Event 7 Flag

enumerator kSCTIMER_Event8Flag

Event 8 Flag

enumerator kSCTIMER_Event9Flag

Event 9 Flag

enumerator kSCTIMER_Event10Flag

Event 10 Flag

enumerator kSCTIMER_Event11Flag

Event 11 Flag

enumerator kSCTIMER_Event12Flag

Event 12 Flag

enumerator kSCTIMER_BusErrorLFlag

Bus error due to write when L counter was not halted

enumerator kSCTIMER_BusErrorHFlag

Bus error due to write when H counter was not halted

typedef enum _sctimer_pwm_mode sctimer_pwm_mode_t

SCTimer PWM operation modes.

`typedef enum _sctimer_counter` `sctimer_counter_t`

SCTimer counters type.

`typedef enum _sctimer_input` `sctimer_input_t`

List of SCTimer input pins.

`typedef enum _sctimer_out` `sctimer_out_t`

List of SCTimer output pins.

`typedef enum _sctimer_pwm_level_select` `sctimer_pwm_level_select_t`

SCTimer PWM output pulse mode: high-true, low-true or no output.

`typedef struct _sctimer_pwm_signal_param` `sctimer_pwm_signal_param_t`

Options to configure a SCTimer PWM signal.

`typedef enum _sctimer_clock_mode` `sctimer_clock_mode_t`

SCTimer clock mode options.

`typedef enum _sctimer_clock_select` `sctimer_clock_select_t`

SCTimer clock select options.

`typedef enum _sctimer_conflict_resolution` `sctimer_conflict_resolution_t`

SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

`typedef enum _sctimer_event_active_direction` `sctimer_event_active_direction_t`

List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

`typedef enum _sctimer_event` `sctimer_event_t`

List of SCTimer event types.

`typedef void (*sctimer_event_callback_t)(void)`

SCTimer callback typedef.

`typedef enum _sctimer_interrupt_enable` `sctimer_interrupt_enable_t`

List of SCTimer interrupts.

`typedef enum _sctimer_status_flags` `sctimer_status_flags_t`

List of SCTimer flags.

`typedef struct _sctimer_config` `sctimer_config_t`

SCTimer configuration structure.

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the `SCTMR_GetDefaultConfig()` function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

`SCT_EV_STATE_STATEMSKn(x)`

`struct _sctimer_pwm_signal_param`

`#include <fsl_sctimer.h>` Options to configure a SCTimer PWM signal.

Public Members

`sctimer_out_t` output

The output pin to use to generate the PWM signal

sctimer_pwm_level_select_t level

PWM output active level select.

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).

struct *_sctimer_config*

#include <fsl_sctimer.h> SCTimer configuration structure.

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the SCTMR_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

bool enableCounterUnify

true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters. User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

sctimer_clock_mode_t clockMode

SCT clock mode value

sctimer_clock_select_t clockSelect

SCT clock select value

bool enableBidirection_l

true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter

bool enableBidirection_h

true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter. This field is used only if the enableCounterUnify is set to false

uint8_t prescale_l

Prescale value to produce the L or unified counter clock

uint8_t prescale_h

Prescale value to produce the H counter clock. This field is used only if the enableCounterUnify is set to false

uint8_t outInitState

Defines the initial output value

uint8_t inputsync

SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16. it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ~bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNCO (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 « INPUTSYNC2) | (1 « INPUTSYNC3);

2.81 SPI: Serial Peripheral Interface Driver

2.82 SPI DMA Driver

```
status_t SPI_MasterTransferCreateHandleDMA(SPI_Type *base, spi_dma_handle_t *handle,  
                                           spi_dma_callback_t callback, void *userData,  
                                           dma_handle_t *txHandle, dma_handle_t  
                                           *rxHandle)
```

Initialize the SPI master DMA handle.

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

Parameters

- base – SPI peripheral base address.
- handle – SPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
- rxHandle – DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

```
status_t SPI_MasterTransferDMA(SPI_Type *base, spi_dma_handle_t *handle, spi_transfer_t  
                               *xfer)
```

Perform a non-blocking SPI transfer using DMA.

Note: This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.
- xfer – Pointer to dma transfer structure.

Return values

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

```
status_t SPI_MasterHalfDuplexTransferDMA(SPI_Type *base, spi_dma_handle_t *handle,  
                                          spi_half_duplex_transfer_t *xfer)
```

Transfers a block of data using a DMA method.

This function using polling way to do the first half transimission and using DMA way to do the srcond half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

Parameters

- base – SPI base pointer
- handle – A pointer to the `spi_master_dma_handle_t` structure which stores the transfer state.
- xfer – A pointer to the `spi_half_duplex_transfer_t` structure.

Returns

status of `status_t`.

```
static inline status_t SPI_SlaveTransferCreateHandleDMA(SPI_Type *base, spi_dma_handle_t
                                                    *handle, spi_dma_callback_t callback,
                                                    void *userData, dma_handle_t
                                                    *txHandle, dma_handle_t *rxHandle)
```

Initialize the SPI slave DMA handle.

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

Parameters

- base – SPI peripheral base address.
- handle – SPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
- rxHandle – DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

```
static inline status_t SPI_SlaveTransferDMA(SPI_Type *base, spi_dma_handle_t *handle,
                                           spi_transfer_t *xfer)
```

Perform a non-blocking SPI transfer using DMA.

Note: This interface returned immediately after transfer initiates, users should call `SPI_GetTransferStatus` to poll the transfer status to check whether SPI transfer finished.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.
- xfer – Pointer to dma transfer structure.

Return values

- `kStatus__Success` – Successfully start a transfer.
- `kStatus__InvalidArgument` – Input argument is invalid.
- `kStatus__SPI_Busy` – SPI is not idle, is running another transfer.

```
void SPI_MasterTransferAbortDMA(SPI_Type *base, spi_dma_handle_t *handle)
```

Abort a SPI transfer using DMA.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.

```
status_t SPI_MasterTransferGetCountDMA(SPI_Type *base, spi_dma_handle_t *handle, size_t
                                     *count)
```

Gets the master DMA transferred bytes.

This function gets the master DMA transferred bytes.

Parameters

- base – SPI peripheral base address.
- handle – A pointer to the spi_dma_handle_t structure which stores the transfer state.
- count – A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

```
static inline void SPI_SlaveTransferAbortDMA(SPI_Type *base, spi_dma_handle_t *handle)
```

Abort a SPI transfer using DMA.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.

```
static inline status_t SPI_SlaveTransferGetCountDMA(SPI_Type *base, spi_dma_handle_t
                                                  *handle, size_t *count)
```

Gets the slave DMA transferred bytes.

This function gets the slave DMA transferred bytes.

Parameters

- base – SPI peripheral base address.
- handle – A pointer to the spi_dma_handle_t structure which stores the transfer state.
- count – A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

```
FSL_SPI_DMA_DRIVER_VERSION
```

SPI DMA driver version.

```
typedef struct _spi_dma_handle spi_dma_handle_t
```

```
typedef void (*spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle, status_t status,
void *userData)
```

SPI DMA callback called at the end of transfer.

```
struct _spi_dma_handle
```

#include <fsl_spi_dma.h> SPI DMA transfer handle, users should not touch the content of the handle.

Public Members

SPI_Type *base

SPI base address

volatile bool txInProgress

Send transfer finished

`volatile bool rxInProgress`
Receive transfer finished

`uint8_t bytesPerFrame`
Bytes in a frame for SPI transfer

`uint8_t lastwordBytes`
The Bytes of lastword for master

`uint16_t txDummy`
The dummy data for TX.

`uint32_t lastword`
The last word for master TX.

`dma_handle_t *txHandle`
DMA handler for SPI send

`dma_handle_t *rxHandle`
DMA handler for SPI receive

`spi_dma_callback_t callback`
Callback for SPI DMA transfer

`void *userData`
User Data for SPI DMA callback

`uint32_t state`
Internal state of SPI DMA transfer

`size_t transferSize`
Bytes need to be transfer

`uint32_t instance`
Index of SPI instance

`const uint8_t *txNextData`
The pointer of next time tx data

`size_t txRemainingBytes`
lastwordBytes + txRemainingBytes is number of data to be send [in bytes]

`uint8_t *rxNextData`
The pointer of next time rx data

`size_t rxRemainingBytes`
Number of data to be received [in bytes]

`bool isSlave`
SPI work in slave mode.

2.83 SPI Driver

`FSL_SPI_DRIVER_VERSION`
SPI driver version.

`enum __spi_xfer_option`
SPI transfer option.

Values:

enumerator kSPI_FrameDelay

A delay may be inserted, defined in the DLY register.

enumerator kSPI_FrameAssert

SSEL will be deasserted at the end of a transfer

enum _spi_shift_direction

SPI data shifter direction options.

Values:

enumerator kSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kSPI_LsbFirst

Data transfers start with least significant bit.

enum _spi_clock_polarity

SPI clock polarity configuration.

Values:

enumerator kSPI_ClockPolarityActiveHigh

Active-high SPI clock (idles low).

enumerator kSPI_ClockPolarityActiveLow

Active-low SPI clock (idles high).

enum _spi_clock_phase

SPI clock phase configuration.

Values:

enumerator kSPI_ClockPhaseFirstEdge

First edge on SCK occurs at the middle of the first cycle of a data transfer.

enumerator kSPI_ClockPhaseSecondEdge

First edge on SCK occurs at the start of the first cycle of a data transfer.

enum _spi_txfifo_watermark

txFIFO watermark values

Values:

enumerator kSPI_TxFifo0

SPI tx watermark is empty

enumerator kSPI_TxFifo1

SPI tx watermark at 1 item

enumerator kSPI_TxFifo2

SPI tx watermark at 2 items

enumerator kSPI_TxFifo3

SPI tx watermark at 3 items

enumerator kSPI_TxFifo4

SPI tx watermark at 4 items

enumerator kSPI_TxFifo5

SPI tx watermark at 5 items

enumerator kSPI_TxFifo6

SPI tx watermark at 6 items

enumerator kSPI_TxFifo7
SPI tx watermark at 7 items

enum _spi_rxfifo_watermark
rxFIFO watermark values

Values:

enumerator kSPI_RxFifo1
SPI rx watermark at 1 item

enumerator kSPI_RxFifo2
SPI rx watermark at 2 items

enumerator kSPI_RxFifo3
SPI rx watermark at 3 items

enumerator kSPI_RxFifo4
SPI rx watermark at 4 items

enumerator kSPI_RxFifo5
SPI rx watermark at 5 items

enumerator kSPI_RxFifo6
SPI rx watermark at 6 items

enumerator kSPI_RxFifo7
SPI rx watermark at 7 items

enumerator kSPI_RxFifo8
SPI rx watermark at 8 items

enum _spi_data_width
Transfer data width.

Values:

enumerator kSPI_Data4Bits
4 bits data width

enumerator kSPI_Data5Bits
5 bits data width

enumerator kSPI_Data6Bits
6 bits data width

enumerator kSPI_Data7Bits
7 bits data width

enumerator kSPI_Data8Bits
8 bits data width

enumerator kSPI_Data9Bits
9 bits data width

enumerator kSPI_Data10Bits
10 bits data width

enumerator kSPI_Data11Bits
11 bits data width

enumerator kSPI_Data12Bits
12 bits data width

enumerator kSPI_Data13Bits

13 bits data width

enumerator kSPI_Data14Bits

14 bits data width

enumerator kSPI_Data15Bits

15 bits data width

enumerator kSPI_Data16Bits

16 bits data width

enum _spi_ssel

Slave select.

Values:

enumerator kSPI_Ssel0

Slave select 0

enumerator kSPI_Ssel1

Slave select 1

enumerator kSPI_Ssel2

Slave select 2

enumerator kSPI_Ssel3

Slave select 3

enum _spi_spol

ssel polarity

Values:

enumerator kSPI_Spol0ActiveHigh

enumerator kSPI_Spol1ActiveHigh

enumerator kSPI_Spol3ActiveHigh

enumerator kSPI_SpolActiveAllHigh

enumerator kSPI_SpolActiveAllLow

SPI transfer status.

Values:

enumerator kStatus_SPI_Busy

SPI bus is busy

enumerator kStatus_SPI_Idle

SPI is idle

enumerator kStatus_SPI_Error

SPI error

enumerator kStatus_SPI_BaudrateNotSupport

Baudrate is not support in current clock source

enumerator kStatus_SPI_Timeout

SPI timeout polling status flags.

enum *_spi_interrupt_enable*

SPI interrupt sources.

Values:

enumerator *kSPI_RxLvlIrq*

Rx level interrupt

enumerator *kSPI_TxLvlIrq*

Tx level interrupt

enum *_spi_statusflags*

SPI status flags.

Values:

enumerator *kSPI_TxEmptyFlag*

txFifo is empty

enumerator *kSPI_TxNotFullFlag*

txFifo is not full

enumerator *kSPI_RxNotEmptyFlag*

rxFIFO is not empty

enumerator *kSPI_RxFullFlag*

rxFIFO is full

typedef enum *_spi_xfer_option* *spi_xfer_option_t*

SPI transfer option.

typedef enum *_spi_shift_direction* *spi_shift_direction_t*

SPI data shifter direction options.

typedef enum *_spi_clock_polarity* *spi_clock_polarity_t*

SPI clock polarity configuration.

typedef enum *_spi_clock_phase* *spi_clock_phase_t*

SPI clock phase configuration.

typedef enum *_spi_txfifo_watermark* *spi_txfifo_watermark_t*

txFIFO watermark values

typedef enum *_spi_rxfifo_watermark* *spi_rxfifo_watermark_t*

rxFIFO watermark values

typedef enum *_spi_data_width* *spi_data_width_t*

Transfer data width.

typedef enum *_spi_ssel* *spi_ssel_t*

Slave select.

typedef enum *_spi_spol* *spi_spol_t*

ssel polarity

typedef struct *_spi_delay_config* *spi_delay_config_t*

SPI delay time configure structure. Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maximum value of these delay time is 15.

typedef struct *_spi_master_config* *spi_master_config_t*

SPI master user configure structure.

```
typedef struct _spi_slave_config spi_slave_config_t
```

SPI slave user configure structure.

```
typedef struct _spi_transfer spi_transfer_t
```

SPI transfer structure.

```
typedef struct _spi_half_duplex_transfer spi_half_duplex_transfer_t
```

SPI half-duplex(master only) transfer structure.

```
typedef struct _spi_config spi_config_t
```

Internal configuration structure used in 'spi' and 'spi_dma' driver.

```
typedef struct _spi_master_handle spi_master_handle_t
```

Master handle type.

```
typedef spi_master_handle_t spi_slave_handle_t
```

Slave handle type.

```
typedef void (*spi_master_callback_t)(SPI_Type *base, spi_master_handle_t *handle, status_t status, void *userData)
```

SPI master callback for finished transmit.

```
typedef void (*spi_slave_callback_t)(SPI_Type *base, spi_slave_handle_t *handle, status_t status, void *userData)
```

SPI slave callback for finished transmit.

```
typedef void (*flexcomm_spi_master_irq_handler_t)(SPI_Type *base, spi_master_handle_t *handle)
```

Typedef for master interrupt handler.

```
typedef void (*flexcomm_spi_slave_irq_handler_t)(SPI_Type *base, spi_slave_handle_t *handle)
```

Typedef for slave interrupt handler.

```
volatile uint8_t s_dummyData[]
```

SPI default SSEL COUNT.

Global variable for dummy data value setting.

SPI_DUMMYDATA

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

SPI_RETRY_TIMES

Retry times for waiting flag.

SPI_DATA(n)

SPI_CTRLMASK

SPI_ASSERTNUM_SSEL(n)

SPI_DEASSERTNUM_SSEL(n)

SPI_DEASSERT_ALL

SPI_FIFOWR_FLAGS_MASK

SPI_FIFOTRIG_TXLVL_GET(base)

SPI_FIFOTRIG_RXLVL_GET(base)

```
struct _spi_delay_config
```

#include <fsl_spi.h> SPI delay time configure structure. Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maximum value of these delay time is 15.

Public Members

uint8_t preDelay

Delay between SSEL assertion and the beginning of transfer.

uint8_t postDelay

Delay between the end of transfer and SSEL deassertion.

uint8_t frameDelay

Delay between frame to frame.

uint8_t transferDelay

Delay between transfer to transfer.

struct __spi_master_config

#include <fsl_spi.h> SPI master user configure structure.

Public Members

bool enableLoopback

Enable loopback for test purpose

bool enableMaster

Enable SPI at initialization time

spi_clock_polarity_t polarity

Clock polarity

spi_clock_phase_t phase

Clock phase

spi_shift_direction_t direction

MSB or LSB

uint32_t baudRate_Bps

Baud Rate for SPI in Hz

spi_data_width_t dataWidth

Width of the data

spi_ssel_t sselNum

Slave select number

spi_spol_t sselPol

Configure active CS polarity

uint8_t txWatermark

txFIFO watermark

uint8_t rxWatermark

rxFIFO watermark

spi_delay_config_t delayConfig

Delay configuration.

struct __spi_slave_config

#include <fsl_spi.h> SPI slave user configure structure.

Public Members

bool enableSlave
 Enable SPI at initialization time

spi_clock_polarity_t polarity
 Clock polarity

spi_clock_phase_t phase
 Clock phase

spi_shift_direction_t direction
 MSB or LSB

spi_data_width_t dataWidth
 Width of the data

spi_spol_t sselPol
 Configure active CS polarity

uint8_t txWatermark
 txFIFO watermark

uint8_t rxWatermark
 rxFIFO watermark

struct *_spi_transfer*
 #include <fsl_spi.h> SPI transfer structure.

Public Members

const uint8_t *txData
 Send buffer

uint8_t *rxData
 Receive buffer

uint32_t configFlags
 Additional option to control transfer, *spi_xfer_option_t*.

size_t dataSize
 Transfer bytes

struct *_spi_half_duplex_transfer*
 #include <fsl_spi.h> SPI half-duplex(master only) transfer structure.

Public Members

const uint8_t *txData
 Send buffer

uint8_t *rxData
 Receive buffer

size_t txDataSize
 Transfer bytes for transmit

size_t rxDataSize
 Transfer bytes

uint32_t configFlags

Transfer configuration flags, spi_xfer_option_t.

bool isPcsAssertInTransfer

If PCS pin keep assert between transmit and receive. true for assert and false for de-assert.

bool isTransmitFirst

True for transmit first and false for receive first.

struct __spi_config

#include <fsl_spi.h> Internal configuration structure used in 'spi' and 'spi_dma' driver.

struct __spi_master_handle

#include <fsl_spi.h> SPI transfer handle structure.

Public Members

const uint8_t *volatile txData

Transfer buffer

uint8_t *volatile rxData

Receive buffer

volatile size_t txRemainingBytes

Number of data to be transmitted [in bytes]

volatile size_t rxRemainingBytes

Number of data to be received [in bytes]

volatile int8_t toReceiveCount

The number of data expected to receive in data width. Since the received count and sent count should be the same to complete the transfer, if the sent count is x and the received count is y, toReceiveCount is x-y.

size_t totalByteCount

A number of transfer bytes

volatile uint32_t state

SPI internal state

spi_master_callback_t callback

SPI callback

void *userData

Callback parameter

uint8_t dataWidth

Width of the data [Valid values: 1 to 16]

uint8_t sselNum

Slave select number to be asserted when transferring data [Valid values: 0 to 3]

uint32_t configFlags

Additional option to control transfer

uint8_t txWatermark

txFIFO watermark

uint8_t rxWatermark

rxFIFO watermark

2.84 SYSTCTL: I2S bridging and signal sharing Configuration

void SYSTCTL_Init(SYSTCTL_Type *base)
SYSTCTL initial.

Parameters

- base – Base address of the SYSTCTL peripheral.

void SYSTCTL_Deinit(SYSTCTL_Type *base)
SYSTCTL deinit.

Parameters

- base – Base address of the SYSTCTL peripheral.

void SYSTCTL_SetFlexcommShareSet(SYSTCTL_Type *base, uint32_t flexCommIndex, uint32_t sckSet, uint32_t wsSet, uint32_t dataInSet, uint32_t dataOutSet)

SYSTCTL share set configure for flexcomm.

Parameters

- base – Base address of the SYSTCTL peripheral.
- flexCommIndex – index of flexcomm, reference `_sysctl_share_src`
- sckSet – share set for sck, reference `_sysctl_share_set_index`
- wsSet – share set for ws, reference `_sysctl_share_set_index`
- dataInSet – share set for data in, reference `_sysctl_share_set_index`
- dataOutSet – share set for data out, reference `_sysctl_dataout_mask`

void SYSTCTL_SetShareSet(SYSTCTL_Type *base, uint32_t flexCommIndex, *sysctl_fcctrlsel_signal_t* signal, uint32_t set)

SYSTCTL share set configure for separate signal.

Parameters

- base – Base address of the SYSTCTL peripheral
- flexCommIndex – index of flexcomm, reference `_sysctl_share_src`
- signal – FCCTRLSEL signal shift
- set – share set for sck, reference `_sysctl_share_set_index`

void SYSTCTL_SetShareSetSrc(SYSTCTL_Type *base, uint32_t setIndex, uint32_t sckShareSrc, uint32_t wsShareSrc, uint32_t dataInShareSrc, uint32_t dataOutShareSrc)

SYSTCTL share set source configure.

Parameters

- base – Base address of the SYSTCTL peripheral
- setIndex – index of share set, reference `_sysctl_share_set_index`
- sckShareSrc – sck source for this share set, reference `_sysctl_share_src`
- wsShareSrc – ws source for this share set, reference `_sysctl_share_src`
- dataInShareSrc – data in source for this share set, reference `_sysctl_share_src`
- dataOutShareSrc – data out source for this share set, reference `_sysctl_dataout_mask`

```
void SYSCTL_SetShareSignalSrc(SYSCTL_Type *base, uint32_t setIndex,
                             sysctl_sharedctrlset_signal_t signal, uint32_t shareSrc)
```

SYSCTL sck source configure.

Parameters

- base – Base address of the SYSCTL peripheral
- setIndex – index of share set, reference `_sysctl_share_set_index`
- signal – FCCTRLSEL signal shift
- shareSrc – sck source fro this share set,reference `_sysctl_share_src`

FSL_SYSCTL_DRIVER_VERSION

Group sysctl driver version for SDK.

Version 2.0.5.

```
enum _sysctl_share_set_index
```

SYSCTL share set.

Values:

```
enumerator kSYSCTL_ShareSet0
```

share set 0

```
enumerator kSYSCTL_ShareSet1
```

share set 1

```
enum _sysctl_fcctrlsel_signal
```

SYSCTL flexcomm signal.

Values:

```
enumerator kSYSCTL_FlexcommSignalSCK
```

SCK signal

```
enumerator kSYSCTL_FlexcommSignalWS
```

WS signal

```
enumerator kSYSCTL_FlexcommSignalDataIn
```

Data in signal

```
enumerator kSYSCTL_FlexcommSignalDataOut
```

Data out signal

```
enum _sysctl_share_src
```

SYSCTL flexcomm index.

Values:

```
enumerator kSYSCTL_Flexcomm0
```

share set 0

```
enumerator kSYSCTL_Flexcomm1
```

share set 1

```
enumerator kSYSCTL_Flexcomm2
```

share set 2

```
enumerator kSYSCTL_Flexcomm4
```

share set 4

```
enumerator kSYSCTL_Flexcomm5
```

share set 5

```
    enumerator kSYSCTL_Flexcomm6
        share set 6
    enumerator kSYSCTL_Flexcomm7
        share set 7
enum __sysctl_dataout_mask
    SYSCTL shared data out mask.
    Values:
    enumerator kSYSCTL_Flexcomm0DataOut
        share set 0
    enumerator kSYSCTL_Flexcomm1DataOut
        share set 1
    enumerator kSYSCTL_Flexcomm2DataOut
        share set 2
    enumerator kSYSCTL_Flexcomm4DataOut
        share set 4
    enumerator kSYSCTL_Flexcomm5DataOut
        share set 5
    enumerator kSYSCTL_Flexcomm6DataOut
        share set 6
    enumerator kSYSCTL_Flexcomm7DataOut
        share set 7
enum __sysctl_sharedctrlset_signal
    SYSCTL flexcomm signal.
    Values:
    enumerator kSYSCTL_SharedCtrlSignalSCK
        SCK signal
    enumerator kSYSCTL_SharedCtrlSignalWS
        WS signal
    enumerator kSYSCTL_SharedCtrlSignalDataIn
        Data in signal
    enumerator kSYSCTL_SharedCtrlSignalDataOut
        Data out signal
typedef enum __sysctl_fcctrlsel_signal sysctl_fcctrlsel_signal_t
    SYSCTL flexcomm signal.
typedef enum __sysctl_sharedctrlset_signal sysctl_sharedctrlset_signal_t
    SYSCTL flexcomm signal.
```

2.85 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver

2.86 USART DMA Driver

```
status_t USART_TransferCreateHandleDMA(USART_Type *base, usart_dma_handle_t *handle,  
                                       usart_dma_transfer_callback_t callback, void  
                                       *userData, dma_handle_t *txDmaHandle,  
                                       dma_handle_t *rxDmaHandle)
```

Initializes the USART handle which is used in transactional functions.

Parameters

- base – USART peripheral base address.
- handle – Pointer to *usart_dma_handle_t* structure.
- callback – Callback function.
- userData – User data.
- txDmaHandle – User-requested DMA handle for TX DMA transfer.
- rxDmaHandle – User-requested DMA handle for RX DMA transfer.

```
status_t USART_TransferSendDMA(USART_Type *base, usart_dma_handle_t *handle,  
                               usart_transfer_t *xfer)
```

Sends data using DMA.

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART DMA transfer structure. See *usart_transfer_t*.

Return values

- *kStatus_Success* – if succeed, others failed.
- *kStatus_USART_TxBusy* – Previous transfer on going.
- *kStatus_InvalidArgument* – Invalid argument.

```
status_t USART_TransferReceiveDMA(USART_Type *base, usart_dma_handle_t *handle,  
                                  usart_transfer_t *xfer)
```

Receives data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – USART peripheral base address.
- handle – Pointer to *usart_dma_handle_t* structure.
- xfer – USART DMA transfer structure. See *usart_transfer_t*.

Return values

- *kStatus_Success* – if succeed, others failed.
- *kStatus_USART_RxBusy* – Previous transfer on going.
- *kStatus_InvalidArgument* – Invalid argument.

```
void USART_TransferAbortSendDMA(USART_Type *base, usart_dma_handle_t *handle)
```

Aborts the sent data using DMA.

This function aborts send data using DMA.

Parameters

- base – USART peripheral base address
- handle – Pointer to `usart_dma_handle_t` structure

`void USART__TransferAbortReceiveDMA(USART_Type *base, usart_dma_handle_t *handle)`

Aborts the received data using DMA.

This function aborts the received data using DMA.

Parameters

- base – USART peripheral base address
- handle – Pointer to `usart_dma_handle_t` structure

`status_t USART__TransferGetReceiveCountDMA(USART_Type *base, usart_dma_handle_t *handle, uint32_t *count)`

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – USART peripheral base address.
- handle – USART 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;

`status_t USART__TransferGetSendCountDMA(USART_Type *base, usart_dma_handle_t *handle, uint32_t *count)`

Get the number of bytes that have been sent.

This function gets the number of bytes that have been sent.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- count – Sent bytes count.

Return values

- `kStatus__NoTransferInProgress` – No receive in progress.
- `kStatus__InvalidArgument` – Parameter is invalid.
- `kStatus__Success` – Get successfully through the parameter count;

`FSL_USART_DMA_DRIVER_VERSION`

USART dma driver version.

`typedef struct usart_dma_handle usart_dma_handle_t`

`typedef void (*usart_dma_transfer_callback_t)(USART_Type *base, usart_dma_handle_t *handle, status_t status, void *userData)`

UART transfer callback function.

`struct usart_dma_handle`

`#include <fsl_usart_dma.h>` UART DMA handle.

Public Members

USART_Type *base

UART peripheral base address.

usart_dma_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

dma_handle_t *txDmaHandle

The DMA TX channel used.

dma_handle_t *rxDmaHandle

The DMA RX channel used.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.87 USART Driver

status_t USART_Init(USART_Type *base, const *usart_config_t* *config, uint32_t srcClock_Hz)

Initializes a USART instance with user configuration structure and peripheral clock.

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the USART_GetDefaultConfig() function. Example below shows how to use this API to configure USART.

```
usart_config_t usartConfig;
usartConfig.baudRate_Bps = 115200U;
usartConfig.parityMode = kUSART_ParityDisabled;
usartConfig.stopBitCount = kUSART_OneStopBit;
USART_Init(USART1, &usartConfig, 20000000U);
```

Parameters

- base – USART peripheral base address.
- config – Pointer to user-defined configuration structure.
- srcClock_Hz – USART clock source frequency in HZ.

Return values

- kStatus_USART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_InvalidArgument – USART base address is not valid
- kStatus_Success – Status USART initialize succeed

```
void USART_CalcTimeoutConfig(uint32_t target_us, uint8_t *rxTimeoutPrescaler, uint32_t *rxTimeoutcounter, uint32_t srcClock_Hz)
```

Calculate the USART instance RX timeout prescaler and counter.

This function for calculate the USART RXFIFO timeout config. This function is used to calculate suitable prescaler and counter for target_us.

```
usart_config_t config;  
config.rxWatermark          = kUSART_RxFifo2;  
config.rxTimeout.enable     = true;  
config.rxTimeout.resetCounterOnEmpty = true;  
config.rxTimeout.resetCounterOnReceive = true;  
USART_CalcTimeoutConfig(200U, &config.rxTimeout.prescaler, &config.rxTimeout.counter,  
                        CLOCK_GetFreq(kCLOCK_BusClk));
```

Parameters

- target_us – Time for rx timeout unit us.
- rxTimeoutPrescaler – The prescaler to be setted after function.
- rxTimeoutcounter – The counter to be setted after function.
- srcClock_Hz – The clockSrc for rx timeout.

```
void USART_SetRxTimeoutConfig(USART_Type *base, const usart_rx_timeout_config *config)
```

Sets the USART instance RX timeout config.

This function configures the USART RXFIFO timeout config. This function is used to config the USART RXFIFO timeout config after the USART module is initialized by the USART_Init.

Parameters

- base – USART peripheral base address.
- config – pointer to receive timeout configuration structure.

```
void USART_Deinit(USART_Type *base)
```

Deinitializes a USART instance.

This function waits for TX complete, disables TX and RX, and disables the USART clock.

Parameters

- base – USART peripheral base address.

```
void USART_GetDefaultConfig(usart_config_t *config)
```

Gets the default configuration structure.

This function initializes the USART configuration structure to a default value. The default values are: usartConfig->baudRate_Bps = 115200U; usartConfig->parityMode = kUSART_ParityDisabled; usartConfig->stopBitCount = kUSART_OneStopBit; usartConfig->bitCountPerChar = kUSART_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;

Parameters

- config – Pointer to configuration structure.

```
status_t USART_SetBaudRate(USART_Type *base, uint32_t baudrate_Bps, uint32_t srcClock_Hz)
```

Sets the USART instance baud rate.

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART_Init.

```
USART_SetBaudRate(USART1, 115200U, 20000000U);
```

Parameters

- base – USART peripheral base address.
- baudrate_Bps – USART baudrate to be set.
- srcClock_Hz – USART clock source frequency in HZ.

Return values

- kStatus_USART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – Set baudrate succeed.
- kStatus_InvalidArgument – One or more arguments are invalid.

status_t USART_Enable32kMode(USART_Type *base, uint32_t baudRate_Bps, bool enableMode32k, uint32_t srcClock_Hz)

Enable 32 kHz mode which USART uses clock from the RTC oscillator as the clock source.

Please note that in order to use a 32 kHz clock to operate USART properly, the RTC oscillator and its 32 kHz output must be manually enabled by user, by calling RTC_Init and setting SYSCON_RTCOSCCTRL_EN bit to 1. And in 32kHz clocking mode the USART can only work at 9600 baudrate or at the baudrate that 9600 can evenly divide, eg: 4800, 3200.

Parameters

- base – USART peripheral base address.
- baudRate_Bps – USART baudrate to be set..
- enableMode32k – true is 32k mode, false is normal mode.
- srcClock_Hz – USART clock source frequency in HZ.

Return values

- kStatus_USART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – Set baudrate succeed.
- kStatus_InvalidArgument – One or more arguments are invalid.

void USART_Enable9bitMode(USART_Type *base, bool enable)

Enable 9-bit data mode for USART.

This function set the 9-bit mode for USART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – USART peripheral base address.
- enable – true to enable, false to disable.

static inline void USART_SetMatchAddress(USART_Type *base, uint8_t address)

Set the USART slave address.

This function configures the address for USART module that works as slave in 9-bit data mode. When the address detection is enabled, 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 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 USART 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 – USART peripheral base address.
- address – USART slave address.

static inline void USART__EnableMatchAddress(USART_Type *base, bool match)

Enable the USART match address feature.

Parameters

- base – USART peripheral base address.
- match – true to enable match address, false to disable.

static inline uint32_t USART__GetStatusFlags(USART_Type *base)

Get USART status flags.

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators `_usart_flags`. To check a specific status, compare the return value with enumerators in `_usart_flags`. For example, to check whether the TX is empty:

```
if (kUSART__TxFifoNotFullFlag & USART__GetStatusFlags(USART1))
{
    ...
}
```

Parameters

- base – USART peripheral base address.

Returns

USART status flags which are Ored by the enumerators in the `_usart_flags`.

static inline void USART__ClearStatusFlags(USART_Type *base, uint32_t mask)

Clear USART status flags.

This function clear supported USART status flags. The mask is a logical OR of enumeration members. See `kUSART_AllClearFlags`. For example:

```
USART__ClearStatusFlags(USART1, kUSART__TxError | kUSART__RxError)
```

Parameters

- base – USART peripheral base address.
- mask – status flags to be cleared.

static inline void USART__EnableInterrupts(USART_Type *base, uint32_t mask)

Enables USART interrupts according to the provided mask.

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See `_usart_interrupt_enable`. For example, to enable TX empty interrupt and RX full interrupt:

```
USART__EnableInterrupts(USART1, kUSART__TxLevelInterruptEnable | kUSART__
↪ RxLevelInterruptEnable);
```

Parameters

- base – USART peripheral base address.
- mask – The interrupts to enable. Logical OR of _usart_interrupt_enable.

static inline void USART__DisableInterrupts(USART_Type *base, uint32_t mask)

Disables USART interrupts according to a provided mask.

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _usart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
USART__DisableInterrupts(USART1, kUSART__TxLevelInterruptEnable | kUSART__
↳ RxLevelInterruptEnable);
```

Parameters

- base – USART peripheral base address.
- mask – The interrupts to disable. Logical OR of _usart_interrupt_enable.

static inline uint32_t USART__GetEnabledInterrupts(USART_Type *base)

Returns enabled USART interrupts.

This function returns the enabled USART interrupts.

Parameters

- base – USART peripheral base address.

static inline void USART__EnableTxDMA(USART_Type *base, bool enable)

Enable DMA for Tx.

static inline void USART__EnableRxDMA(USART_Type *base, bool enable)

Enable DMA for Rx.

static inline void USART__EnableCTS(USART_Type *base, bool enable)

Enable CTS. This function will determine whether CTS is used for flow control.

Parameters

- base – USART peripheral base address.
- enable – Enable CTS or not, true for enable and false for disable.

static inline void USART__EnableContinuousSCLK(USART_Type *base, bool enable)

Continuous Clock generation. By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this function, SCLK will run continuously in synchronous mode, allowing characters to be received on Un_RxD independently from transmission on Un_TXD).

Parameters

- base – USART peripheral base address.
- enable – Enable Continuous Clock generation mode or not, true for enable and false for disable.

static inline void USART__EnableAutoClearSCLK(USART_Type *base, bool enable)

Enable Continuous Clock generation bit auto clear. While enable this function, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

- base – USART peripheral base address.
- enable – Enable auto clear or not, true for enable and false for disable.

static inline void USART_SetRxFifoWatermark(USART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – USART peripheral base address.
- water – Rx FIFO watermark.

static inline void USART_SetTxFifoWatermark(USART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – USART peripheral base address.
- water – Tx FIFO watermark.

static inline void USART_WriteByte(USART_Type *base, uint8_t data)

Writes to the FIFOWR register.

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

Parameters

- base – USART peripheral base address.
- data – The byte to write.

static inline uint8_t USART_ReadByte(USART_Type *base)

Reads the FIFORD register directly.

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

Parameters

- base – USART peripheral base address.

Returns

The byte read from USART data register.

static inline uint8_t USART_GetRxFifoCount(USART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – USART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t USART_GetTxFifoCount(USART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – USART peripheral base address.

Returns

tx FIFO data count.

void USART_SendAddress(USART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – USART peripheral base address.
- address – USART slave address.

status_t USART_WriteBlocking(USART_Type *base, const uint8_t *data, size_t length)

Writes to the TX register using a blocking method.

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

- base – USART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_USART_Timeout – Transmission timed out and was aborted.
- kStatus_InvalidArgument – Invalid argument.
- kStatus_Success – Successfully wrote all data.

status_t USART_ReadBlocking(USART_Type *base, uint8_t *data, size_t length)

Read RX data register using a blocking method.

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data and read data from the TX register.

Parameters

- base – USART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_USART_FramingError – Receiver overrun happened while receiving data.
- kStatus_USART_ParityError – Noise error happened while receiving data.
- kStatus_USART_NoiseError – Framing error happened while receiving data.
- kStatus_USART_RxError – Overflow or underflow rxFIFO happened.
- kStatus_USART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t USART_TransferCreateHandle(USART_Type *base, *usart_handle_t* *handle, *usart_transfer_callback_t* callback, void *userData)

Initializes the USART handle.

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

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

```
status_t USART_TransferSendNonBlocking(USART_Type *base, usart_handle_t *handle,  
                                       usart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the IRQ handler, the USART driver calls the callback function and passes the `kStatus_USART_TxIdle` as status parameter.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART transfer structure. See `usart_transfer_t`.

Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_USART_TxBusy` – Previous transmission still not finished, data not all written to TX register yet.
- `kStatus_InvalidArgument` – Invalid argument.

```
void USART_TransferStartRingBuffer(USART_Type *base, usart_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 USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the `USART_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 the 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 – USART peripheral base address.
- handle – USART handle pointer.
- ringBuffer – Start address of the ring buffer for background receiving. Pass `NULL` to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

```
void USART_TransferStopRingBuffer(USART_Type *base, usart_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 – USART peripheral base address.
- handle – USART handle pointer.

```
size_t USART_TransferGetRxRingBufferLength(usart_handle_t *handle)
```

Get the length of received data in RX ring buffer.

Parameters

- handle – USART handle pointer.

Returns

Length of received data in RX ring buffer.

```
void USART_TransferAbortSend(USART_Type *base, usart_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 still not sent out.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.

```
status_t USART_TransferGetSendCount(USART_Type *base, usart_handle_t *handle, uint32_t *count)
```

Get 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 interrupt method.

Parameters

- base – USART peripheral base address.
- handle – USART 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 USART_TransferReceiveNonBlocking(USART_Type *base, usart_handle_t *handle, usart_transfer_t *xfer, size_t *receivedBytes)
```

Receives a buffer of data using an interrupt method.

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_USART_RxIdle`. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the `xfer->data` and this function returns with the parameter `receivedBytes` set to 5. For the left 5 bytes, newly arrived data is saved from the `xfer->data[5]`. When 5 bytes are received, the USART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the `xfer->data`. When all data is received, the upper layer is notified.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART transfer structure, see `usart_transfer_t`.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into transmit queue.
- kStatus_USART_RxBusy – Previous receive request is not finished.

- `kStatus_InvalidArgument` – Invalid argument.

`void USART_TransferAbortReceive(USART_Type *base, usart_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` – USART peripheral base address.
- `handle` – USART handle pointer.

`status_t USART_TransferGetReceiveCount(USART_Type *base, usart_handle_t *handle, uint32_t *count)`

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- `base` – USART peripheral base address.
- `handle` – USART 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 USART_TransferHandleIRQ(USART_Type *base, usart_handle_t *handle)`

USART IRQ handle function.

This function handles the USART transmit and receive IRQ request.

Parameters

- `base` – USART peripheral base address.
- `handle` – USART handle pointer.

`FSL_USART_DRIVER_VERSION`

USART driver version.

Error codes for the USART driver.

Values:

enumerator `kStatus_USART_TxBusy`
Transmitter is busy.

enumerator `kStatus_USART_RxBusy`
Receiver is busy.

enumerator `kStatus_USART_TxIdle`
USART transmitter is idle.

enumerator `kStatus_USART_RxIdle`
USART receiver is idle.

enumerator `kStatus_USART_TxError`
Error happens on txFIFO.

enumerator kStatus_USART_RxError

Error happens on rxFIFO.

enumerator kStatus_USART_RxRingBufferOverflow

Error happens on rx ring buffer

enumerator kStatus_USART_NoiseError

USART noise error.

enumerator kStatus_USART_FramingError

USART framing error.

enumerator kStatus_USART_ParityError

USART parity error.

enumerator kStatus_USART_BaudrateNotSupport

Baudrate is not support in current clock source

enum __usart_sync_mode

USART synchronous mode.

Values:

enumerator kUSART_SyncModeDisabled

Asynchronous mode.

enumerator kUSART_SyncModeSlave

Synchronous slave mode.

enumerator kUSART_SyncModeMaster

Synchronous master mode.

enum __usart_parity_mode

USART parity mode.

Values:

enumerator kUSART_ParityDisabled

Parity disabled

enumerator kUSART_ParityEven

Parity enabled, type even, bit setting: PE | PT = 10

enumerator kUSART_ParityOdd

Parity enabled, type odd, bit setting: PE | PT = 11

enum __usart_stop_bit_count

USART stop bit count.

Values:

enumerator kUSART_OneStopBit

One stop bit

enumerator kUSART_TwoStopBit

Two stop bits

enum __usart_data_len

USART data size.

Values:

enumerator kUSART_7BitsPerChar

Seven bit mode

enumerator kUSART_8BitsPerChar

Eight bit mode

enum _usart_clock_polarity

USART clock polarity configuration, used in sync mode.

Values:

enumerator kUSART_RxSampleOnFallingEdge

Un_RXD is sampled on the falling edge of SCLK.

enumerator kUSART_RxSampleOnRisingEdge

Un_RXD is sampled on the rising edge of SCLK.

enum _usart_txfifo_watermark

txFIFO watermark values

Values:

enumerator kUSART_TxFifo0

USART tx watermark is empty

enumerator kUSART_TxFifo1

USART tx watermark at 1 item

enumerator kUSART_TxFifo2

USART tx watermark at 2 items

enumerator kUSART_TxFifo3

USART tx watermark at 3 items

enumerator kUSART_TxFifo4

USART tx watermark at 4 items

enumerator kUSART_TxFifo5

USART tx watermark at 5 items

enumerator kUSART_TxFifo6

USART tx watermark at 6 items

enumerator kUSART_TxFifo7

USART tx watermark at 7 items

enum _usart_rxfifo_watermark

rxFIFO watermark values

Values:

enumerator kUSART_RxFifo1

USART rx watermark at 1 item

enumerator kUSART_RxFifo2

USART rx watermark at 2 items

enumerator kUSART_RxFifo3

USART rx watermark at 3 items

enumerator kUSART_RxFifo4

USART rx watermark at 4 items

enumerator kUSART_RxFifo5

USART rx watermark at 5 items

enumerator kUSART_RxFifo6
USART rx watermark at 6 items

enumerator kUSART_RxFifo7
USART rx watermark at 7 items

enumerator kUSART_RxFifo8
USART rx watermark at 8 items

enum __usart_interrupt_enable
USART interrupt configuration structure, default settings all disabled.

Values:

enumerator kUSART_TxErrorInterruptEnable

enumerator kUSART_RxErrorInterruptEnable

enumerator kUSART_TxLevelInterruptEnable

enumerator kUSART_RxLevelInterruptEnable

enumerator kUSART_TxIdleInterruptEnable
Transmitter idle.

enumerator kUSART_CtsChangeInterruptEnable
Change in the state of the CTS input.

enumerator kUSART_RxBreakChangeInterruptEnable
Break condition asserted or deasserted.

enumerator kUSART_RxStartInterruptEnable
Rx start bit detected.

enumerator kUSART_FramingErrorInterruptEnable
Framing error detected.

enumerator kUSART_ParityErrorInterruptEnable
Parity error detected.

enumerator kUSART_NoiseErrorInterruptEnable
Noise error detected.

enumerator kUSART_AutoBaudErrorInterruptEnable
Auto baudrate error detected.

enumerator kUSART_RxTimeoutInterruptEnable
Receive timeout detected.

enumerator kUSART_AllInterruptEnables

enum __usart_flags
USART status flags.
This provides constants for the USART status flags for use in the USART functions.

Values:

enumerator kUSART_TxError
TXERR bit, sets if TX buffer is error

enumerator kUSART_RxError
RXERR bit, sets if RX buffer is error

enumerator `kUSART_TxFifoEmptyFlag`
TXEMPTY bit, sets if TX buffer is empty

enumerator `kUSART_TxFifoNotFullFlag`
TXNOTFULL bit, sets if TX buffer is not full

enumerator `kUSART_RxFifoNotEmptyFlag`
RXNOEMPTY bit, sets if RX buffer is not empty

enumerator `kUSART_RxFifoFullFlag`
RXFULL bit, sets if RX buffer is full

enumerator `kUSART_RxIdleFlag`
Receiver idle.

enumerator `kUSART_TxIdleFlag`
Transmitter idle.

enumerator `kUSART_CtsAssertFlag`
CTS signal high.

enumerator `kUSART_CtsChangeFlag`
CTS signal changed interrupt status.

enumerator `kUSART_BreakDetectFlag`
Break detected. Self cleared when rx pin goes high again.

enumerator `kUSART_BreakDetectChangeFlag`
Break detect change interrupt flag. A change in the state of receiver break detection.

enumerator `kUSART_RxStartFlag`
Rx start bit detected interrupt flag.

enumerator `kUSART_FramingErrorFlag`
Framing error interrupt flag.

enumerator `kUSART_ParityErrorFlag`
parity error interrupt flag.

enumerator `kUSART_NoiseErrorFlag`
Noise error interrupt flag.

enumerator `kUSART_AutobaudErrorFlag`
Auto baudrate error interrupt flag, caused by the baudrate counter timeout before the end of start bit.

enumerator `kUSART_RxTimeoutFlag`
RXTIMEOUT bit, sets if RX FIFO Timeout.

enumerator `kUSART_AllClearFlags`

typedef enum *usart_sync_mode* `usart_sync_mode_t`
USART synchronous mode.

typedef enum *usart_parity_mode* `usart_parity_mode_t`
USART parity mode.

typedef enum *usart_stop_bit_count* `usart_stop_bit_count_t`
USART stop bit count.

typedef enum *usart_data_len* `usart_data_len_t`
USART data size.

`typedef enum _usart_clock_polarity` `usart_clock_polarity_t`
USART clock polarity configuration, used in sync mode.

`typedef enum _usart_txfifo_watermark` `usart_txfifo_watermark_t`
txFIFO watermark values

`typedef enum _usart_rxfifo_watermark` `usart_rxfifo_watermark_t`
rxFIFO watermark values

`typedef struct _usart_rx_timeout_config` `usart_rx_timeout_config`
USART receive timeout configuration structure.

`typedef struct _usart_config` `usart_config_t`
USART configuration structure.

`typedef struct _usart_transfer` `usart_transfer_t`
USART transfer structure.

`typedef struct _usart_handle` `usart_handle_t`

`typedef void (*usart_transfer_callback_t)(USART_Type *base, usart_handle_t *handle, status_t status, void *userData)`
USART transfer callback function.

`typedef void (*flexcomm_usart_irq_handler_t)(USART_Type *base, usart_handle_t *handle)`
Typedef for usart interrupt handler.

`uint32_t` `USART_GetInstance(USART_Type *base)`
Returns instance number for USART peripheral base address.

`USART_FIFOTRIG_TXLVL_GET(base)`

`USART_FIFOTRIG_RXLVL_GET(base)`

`UART_RETRY_TIMES`
Retry times for waiting flag.

Defining to zero means to keep waiting for the flag until it is assert/deassert in blocking transfer, otherwise the program will wait until the `UART_RETRY_TIMES` counts down to 0, if the flag still remains unchanged then program will return `kStatus_USART_Timeout`. It is not advised to use this macro in formal application to prevent any hardware error because the actual wait period is affected by the compiler and optimization.

`struct _usart_rx_timeout_config`
`#include <fsl_usart.h>` USART receive timeout configuration structure.

Public Members

`bool` `enable`
Enable RX timeout

`bool` `resetCounterOnEmpty`
Enable RX timeout counter reset when RX FIFO becomes empty.

`bool` `resetCounterOnReceive`
Enable RX timeout counter reset when RX FIFO receives data from the transmitter side.

`uint32_t` `counter`
RX timeout counter

`uint8_t` `prescaler`
RX timeout prescaler

```
struct __usart_config
    #include <fsl_usart.h> USART configuration structure.
```

Public Members

```
uint32_t baudRate_Bps
    USART baud rate

usart_parity_mode_t parityMode
    Parity mode, disabled (default), even, odd

usart_stop_bit_count_t stopBitCount
    Number of stop bits, 1 stop bit (default) or 2 stop bits

usart_data_len_t bitCountPerChar
    Data length - 7 bit, 8 bit

bool loopback
    Enable peripheral loopback

bool enableRx
    Enable RX

bool enableTx
    Enable TX

bool enableContinuousSCLK
    USART continuous Clock generation enable in synchronous master mode.

bool enableMode32k
    USART uses 32 kHz clock from the RTC oscillator as the clock source.

bool enableHardwareFlowControl
    Enable hardware control RTS/CTS

usart_txfifo_watermark_t txWatermark
    txFIFO watermark

usart_rxfifo_watermark_t rxWatermark
    rxFIFO watermark

usart_sync_mode_t syncMode
    Transfer mode select - asynchronous, synchronous master, synchronous slave.

usart_clock_polarity_t clockPolarity
    Selects the clock polarity and sampling edge in synchronous mode.

usart_rx_timeout_config rxTimeout
    rx timeout configuration
```

```
struct __usart_transfer
    #include <fsl_usart.h> USART transfer structure.
```

Public Members

```
size_t dataSize
    The byte count to be transfer.
```

```
struct __usart_handle
    #include <fsl_usart.h> USART handle structure.
```


Public Members

`const uint8_t *volatile txData`
Address of remaining data to send.

`volatile size_t txDataSize`
Size of the remaining data to send.

`size_t txDataSizeAll`
Size of the data to send out.

`uint8_t *volatile rxData`
Address of remaining data to receive.

`volatile size_t rxDataSize`
Size of the remaining data to receive.

`size_t rxDataSizeAll`
Size of the data to receive.

`uint8_t *rxRingBuffer`
Start address of the receiver ring buffer.

`size_t rxRingBufferSize`
Size of the ring buffer.

`volatile uint16_t rxRingBufferHead`
Index for the driver to store received data into ring buffer.

`volatile uint16_t rxRingBufferTail`
Index for the user to get data from the ring buffer.

`usart_transfer_callback_t callback`
Callback function.

`void *userData`
USART callback function parameter.

`volatile uint8_t txState`
TX transfer state.

`volatile uint8_t rxState`
RX transfer state

`uint8_t txWatermark`
txFIFO watermark

`uint8_t rxWatermark`
rxFIFO watermark

`union __unnamed77__`

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.

2.88 UTICK: Mictotick Timer Driver

`void UTICK_Init(UTICK_Type *base)`

Initializes an UTICK by turning its bus clock on.

`void UTICK_Deinit(UTICK_Type *base)`

Deinitializes a UTICK instance.

This function shuts down Utick bus clock

Parameters

- `base` – UTICK peripheral base address.

`uint32_t UTICK_GetStatusFlags(UTICK_Type *base)`

Get Status Flags.

This returns the status flag

Parameters

- `base` – UTICK peripheral base address.

Returns

status register value

`void UTICK_ClearStatusFlags(UTICK_Type *base)`

Clear Status Interrupt Flags.

This clears intr status flag

Parameters

- `base` – UTICK peripheral base address.

Returns

none

`void UTICK_SetTick(UTICK_Type *base, utick_mode_t mode, uint32_t count, utick_callback_t cb)`

Starts UTICK.

This function starts a repeat/onetime countdown with an optional callback

Parameters

- `base` – UTICK peripheral base address.
- `mode` – UTICK timer mode (ie `kUTICK_onetime` or `kUTICK_repeat`)
- `count` – UTICK timer mode (ie `kUTICK_onetime` or `kUTICK_repeat`)
- `cb` – UTICK callback (can be left as `NULL` if none, otherwise should be a `void func(void)`)

Returns

none

`void UTICK_HandleIRQ(UTICK_Type *base, utick_callback_t cb)`

UTICK Interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in `UTICK_SetTick()`). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- `base` – UTICK peripheral base address.

- `cb` – callback scheduled for this instance of UTICK

Returns

none

FSL_UTICK_DRIVER_VERSION

UTICK driver version 2.0.5.

enum `_utick_mode`

UTICK timer operational mode.

*Values:*enumerator `kUTICK_Onetime`

Trigger once

enumerator `kUTICK_Repeat`

Trigger repeatedly

typedef enum `_utick_mode` `utick_mode_t`

UTICK timer operational mode.

typedef void (`*utick_callback_t`)(void)

UTICK callback function.

2.89 VREF: Voltage Reference Driver

void `VREF_Init(VREF_Type *base, const vref_config_t *config)`

Enables the clock gate and configures the VREF module according to the configuration structure.

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up `vref_config_t` parameters and how to call the `VREF_Init` function by passing in these parameters.

```
vref_config_t vrefConfig;
VREF_GetDefaultConfig(VREF, &vrefConfig);
vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
VREF_Init(VREF, &vrefConfig);
```

Parameters

- `base` – VREF peripheral address.
- `config` – Pointer to the configuration structure.

void `VREF_Deinit(VREF_Type *base)`

Stops and disables the clock for the VREF module.

This function should be called to shut down the module. This is an example.

```
vref_config_t vrefUserConfig;
VREF_GetDefaultConfig(VREF, &vrefUserConfig);
VREF_Init(VREF, &vrefUserConfig);
...
VREF_Deinit(VREF);
```

Parameters

- `base` – VREF peripheral address.

`void VREF_GetDefaultConfig(vref_config_t *config)`

Initializes the VREF configuration structure.

This function initializes the VREF configuration structure to default values. This is an example.

```
config->bufferMode = kVREF_ModeHighPowerBuffer;
config->enableInternalVoltageRegulator = true;
config->enableChopOscillator           = true;
config->enableHCBandgap                = true;
config->enableCurvatureCompensation    = true;
config->enableLowPowerBuff             = true;
```

Parameters

- `config` – Pointer to the initialization structure.

`void VREF_SetVrefTrimVal(VREF_Type *base, uint8_t trimValue)`

Sets a TRIM value for the accurate 1.0V bandgap output.

This function sets a TRIM value for the reference voltage. It will trim the accurate 1.0V bandgap by 0.5mV each step.

Parameters

- `base` – VREF peripheral address.
- `trimValue` – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

`void VREF_SetTrim21Val(VREF_Type *base, uint8_t trim21Value)`

Sets a TRIM value for the accurate buffered VREF output.

This function sets a TRIM value for the reference voltage. If buffer mode be set to other values (Buf21 enabled), it will trim the VREF_OUT by 0.1V each step from 1.0V to 2.1V.

Note: When Buf21 is enabled, the value of UTRIM[TRIM2V1] should be ranged from 0b0000 to 0b1011 in order to trim the output voltage from 1.0V to 2.1V, other values will make the VREF_OUT to default value, 1.0V.

Parameters

- `base` – VREF peripheral address.
- `trim21Value` – Value of the trim register to set the output reference voltage (maximum 0xF (4-bit)).

`uint8_t VREF_GetVrefTrimVal(VREF_Type *base)`

Reads the trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[VREFTRIM] (13:8)

Parameters

- `base` – VREF peripheral address.

Returns

6-bit value of trim setting.

`uint8_t VREF_GetTrim21Val(VREF_Type *base)`

Reads the VREF 2.1V trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[TRIM2V1] (3:0),

Parameters

- base – VREF peripheral address.

Returns

4-bit value of trim setting.

FSL_VREF_DRIVER_VERSION

Version 2.4.0.

enum _vref_buffer_mode

VREF buffer modes.

Values:

enumerator kVREF_ModeBandgapOnly

Bandgap enabled/standby.

enumerator kVREF_ModeLowPowerBuffer

Low-power buffer mode enabled

enumerator kVREF_ModeHighPowerBuffer

High-power buffer mode enabled

typedef enum _vref_buffer_mode vref_buffer_mode_t

VREF buffer modes.

typedef struct _vref_config vref_config_t

The description structure for the VREF module.

struct _vref_config

#include <fsl_vref.h> The description structure for the VREF module.

Public Members

vref_buffer_mode_t bufferMode

Buffer mode selection

bool enableInternalVoltageRegulator

Provide additional supply noise rejection.

bool enableChopOscillator

Enable Chop oscillator.

bool enableHCBandgap

Enable High-Accurate bandgap.

bool enableCurvatureCompensation

Enable second order curvature compensation.

bool enableLowPowerBuff

Provides bias current for other peripherals.

2.90 WWDT: Windowed Watchdog Timer Driver

void WWDT_GetDefaultConfig(*wwdt_config_t* *config)

Initializes WWDT configure structure.

This function initializes the WWDT configure structure to default value. The default value are:

```
config->enableWwdt = true;
config->enableWatchdogReset = false;
config->enableWatchdogProtect = false;
config->enableLockOscillator = false;
config->windowValue = 0xFFFFFfU;
config->timeoutValue = 0xFFFFFfU;
config->warningValue = 0;
```

See also:

wwdt_config_t

Parameters

- config – Pointer to WWDT config structure.

void WWDT_Init(WWDT_Type *base, const wwdt_config_t *config)

Initializes the WWDT.

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
wwdt_config_t config;
WWDT_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
WWDT_Init(wwdt_base,&config);
```

Parameters

- base – WWDT peripheral base address
- config – The configuration of WWDT

void WWDT_Deinit(WWDT_Type *base)

Shuts down the WWDT.

This function shuts down the WWDT.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Enable(WWDT_Type *base)

Enables the WWDT module.

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Disable(WWDT_Type *base)

Disables the WWDT module.

Deprecated:

Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

- base – WWDT peripheral base address

static inline uint32_t WWDT_GetStatusFlags(WWDT_Type *base)

Gets all WWDT status flags.

This function gets all status flags.

Example for getting Timeout Flag:

```
uint32_t status;
status = WWDT_GetStatusFlags(wwdt_base) & kWWDT_TimeoutFlag;
```

Parameters

- base – WWDT peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `_wwdt_status_flags_t`

void WWDT_ClearStatusFlags(WWDT_Type *base, uint32_t mask)

Clear WWDT flag.

This function clears WWDT status flag.

Example for clearing warning flag:

```
WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
```

Parameters

- base – WWDT peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `_wwdt_status_flags_t`

static inline void WWDT_SetWarningValue(WWDT_Type *base, uint32_t warningValue)

Set the WWDT warning value.

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

- base – WWDT peripheral base address
- warningValue – WWDT warning value.

static inline void WWDT_SetTimeoutValue(WWDT_Type *base, uint32_t timeoutCount)

Set the WWDT timeout value.

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is $TWDCLK * 256 * 4$. If `enableWatchdogProtect` flag is true in `wwdt_config_t` config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

- base – WWDT peripheral base address
- timeoutCount – WWDT timeout value, count of WWDT clock tick.

static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)

Sets the WWDT window value.

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

- base – WWDT peripheral base address
- windowValue – WWDT window value.

void WWDT_Refresh(WWDT_Type *base)

Refreshes the WWDT timer.

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WWDT peripheral base address

FSL_WWDT_DRIVER_VERSION

Defines WWDT driver version.

WWDT_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WWDT_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

enum _wwdt_status_flags_t

WWDT status flags.

This structure contains the WWDT status flags for use in the WWDT functions.

Values:

enumerator kWWDT_TimeoutFlag

Time-out flag, set when the timer times out

enumerator kWWDT_WarningFlag

Warning interrupt flag, set when timer is below the value WDWARNINT

typedef struct _wwdt_config wwdt_config_t

Describes WWDT configuration structure.

struct _wwdt_config

#include <fsl_wwdt.h> Describes WWDT configuration structure.

Public Members

bool enableWwdt

Enables or disables WWDT

bool enableWatchdogReset

true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset

`bool enableWatchdogProtect`

 true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time

`bool enableLockOscillator`

 true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator

`uint32_t windowValue`

 Window value, set this to 0xFFFFFFFF if windowing is not in effect

`uint32_t timeoutValue`

 Timeout value

`uint32_t warningValue`

 Watchdog time counter value that will generate a warning interrupt. Set this to 0 for no warning

`uint32_t clockFreq_Hz`

 Watchdog clock source frequency.

Chapter 3

Middleware

3.1 Motor Control

3.1.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called “packet-driven BDM” interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT](<https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/>) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to [FreeMASTER community](#) or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the `src/platforms` directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called `FMSTR_TRANSPORT` with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The `mcuxsdk` folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a “middleware” component which may be downloaded along with the example applications from <https://mcuxpresso.nxp.com/en/welcome>.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- [The official FreeMASTER middleware repository.](#)
- [Online version of this document](#)

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer’s physical or virtual COM port. The typical transmission speed is 115200 bps.

- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_pd_bdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER “middleware” driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample applications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.

- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.

- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory

block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- “External” single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- “True” single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as “automatic” in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- **src/platforms** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- **src/common** folder—contains the common driver source files shared by the driver for all supported platforms. All the .c files must be added to the project, compiled, and linked together with the application.
 - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
 - *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
 - *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
 - *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
 - *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
 - *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
 - *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
 - *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
 - *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.

- *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
- *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster_can.h* - defines the low-level message-oriented CAN API.
- *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.
- *freemaster_net.h* - definitions related to the Network transport.
- *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster_utils.h* - definitions related to utility code.
- **src/drivers/[sdk]/serial** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
 - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- **src/drivers/[sdk]/can** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
 - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- **src/drivers/[sdk]/network** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
 - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
 - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR    [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See [Driver interrupt modes](#).

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- FMSTR_SERIAL - serial communication protocol
- FMSTR_CAN - using CAN communication
- FMSTR_PDBDM - using packet-driven BDM communication
- FMSTR_NET - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)
- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as **FMSTR_SERIAL_DRV**. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetSerialBaseAddress()` to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode. The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call `FMSTR_SetCanBaseAddress()` to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with `FMSTR_CAN_EXTID` bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options**FMSTR_DISABLE**

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.
Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options**FMSTR_USE_SCOPE**

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options**FMSTR_USE_RECORDER**

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()' API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()' API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.
Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature. Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options**FMSTR_USE_TSA**

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions. Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used. Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support. The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_QUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial “character time” which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the *FMSTR_Poll* routine. An application interrupt can occur in the middle of the

Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see [Driver interrupt modes](#)), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all *.c files of the FreeMASTER driver from the `src/common/platforms/[your_platform]` folder are a part of the project. See [Driver files](#) for more details.
- Configure the FreeMASTER driver by creating or editing the `freemaster_cfg.h` file and by saving it into the application project directory. See [Driver configuration](#) for more details.
- Include the `freemaster.h` file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the `freemaster_cfg.h` file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: `freemaster.h`
- Implementation: `freemaster_protocol.c`

Description This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the “idle” time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the *FMSTR_Poll* function is called at least once per the time calculated as:

$$N * Tchar$$

where:

- *N* is equal to the length of the receive FIFO queue (configured by the *FMSTR_COMM_RQUEUE_SIZE* macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see [Configurable items](#).

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.

- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINTn	Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)
FMSTR_TSA_SINTn	Signed integer type of size <i>n</i> bits (n=8,16,32,64)
FMSTR_TSA_FRACn	Fractional number of size <i>n</i> bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(<i>m,n</i>)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(<i>m,n</i>)	Unsigned fractional number in general UQ form (m+n total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FM-STR_TSA_USERTYPE(<i>name</i>)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()
```

```
FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmrstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)
```

```
/* Directory entry applies to all subsequent MEMFILE entries */
```

```
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */
```

(continues on next page)

(continued from previous page)

```

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()

```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```

FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪ tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);

```

- Declaration: *freemaster.h*

- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
 - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
 - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
 - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the *FMSTR_USE_TSA_DYNAMIC* configuration option and when the *FMSTR_SetUpTsaBuff* function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the *FMSTR_APPCMDRESULT_NOCMD* constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the *FMSTR_AppCmdAck* call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The *FMSTR_GetAppCmd* function does not report the commands for which a callback handler function exists. If the *FMSTR_GetAppCmd* function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns *FMSTR_APPCMDRESULT_NOCMD*.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_XXX and FMSTR_PIPE_SIZE_XXX constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,  
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the *nGranularity* value equal to the *nLength* value, all data are considered as one chunk which is either written successfully as a whole or not at all. The *nGranularity* value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
<i>FM-STR_ADDR</i> 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.

3.2 Multimedia

3.2.1 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](#)

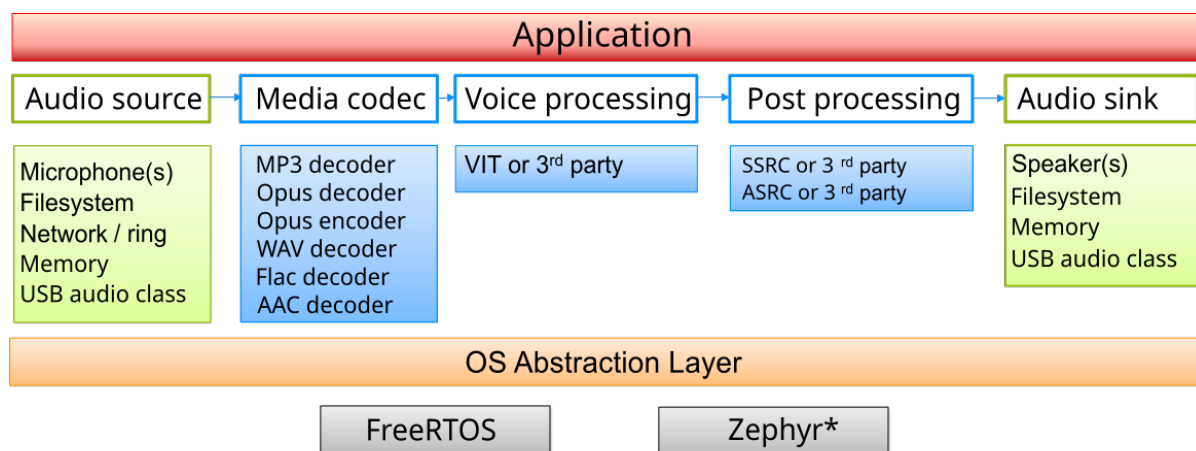
Visit [Maestro - Documentation](#) to review details on the contents in this sub-repo.

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

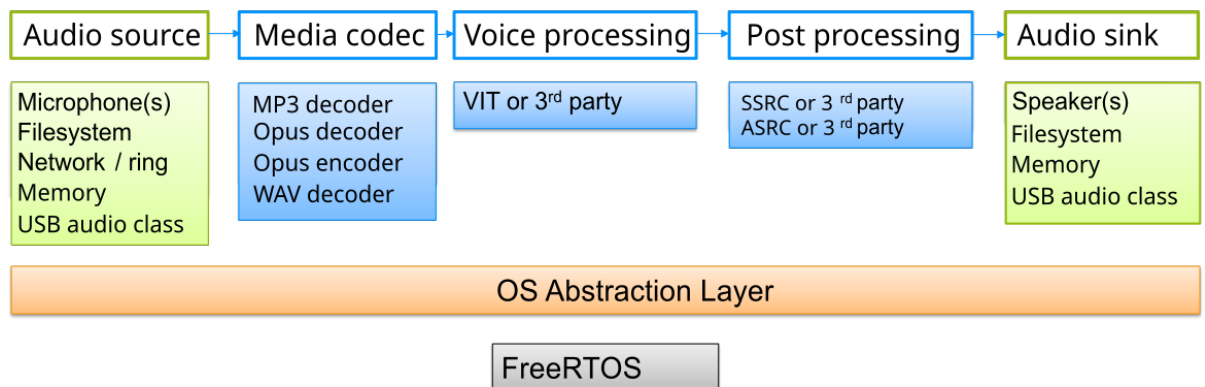
File structure

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.

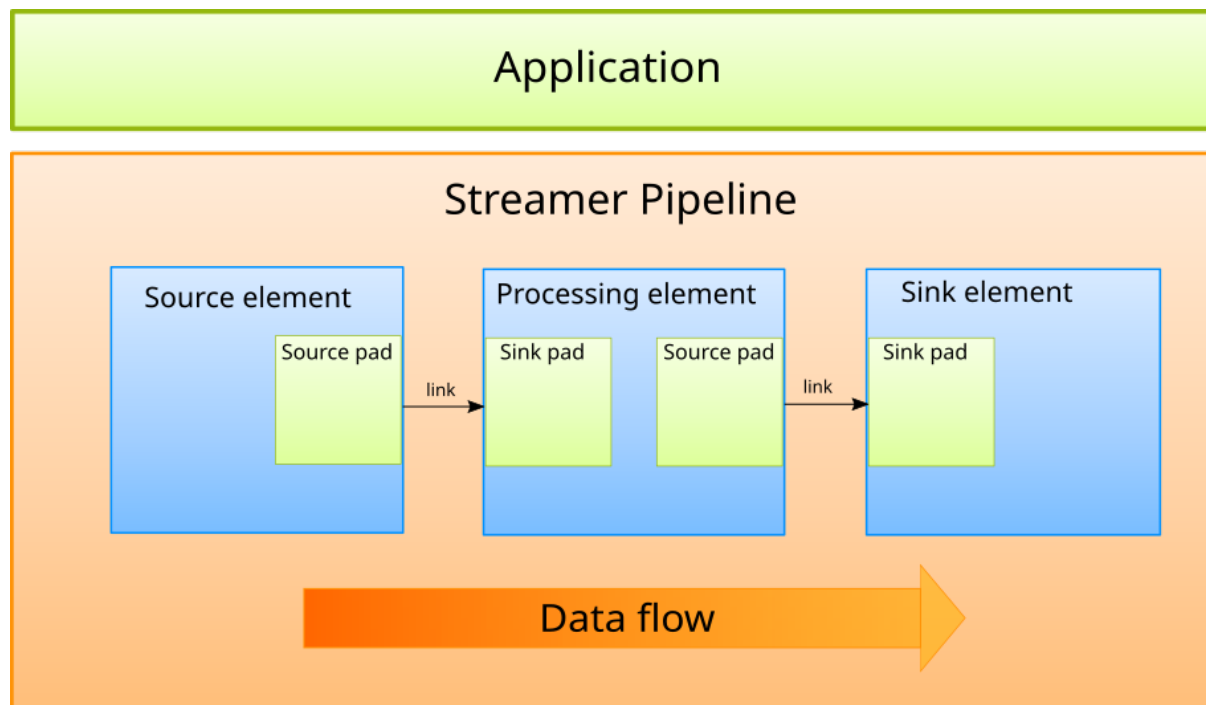
Mostly the user doesn't need to create a new element type, but just create an element index.

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.

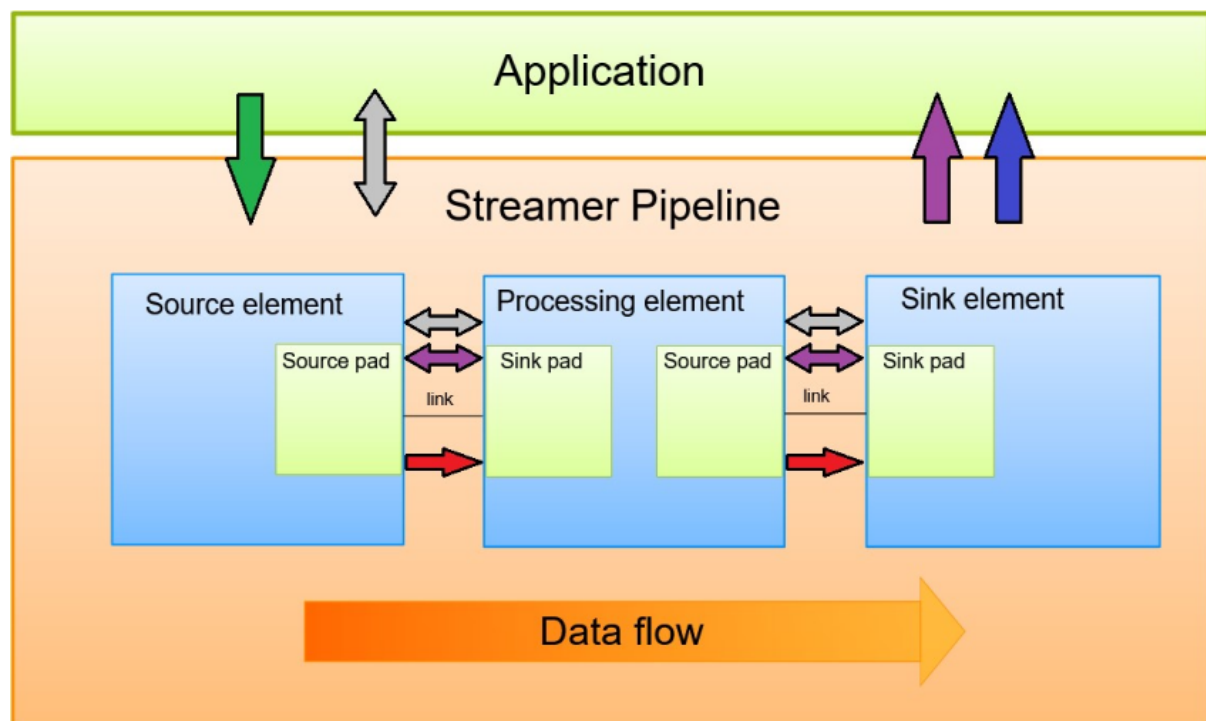
It is important to know that each element type (`StreamElementType`) can be associated with more than one element index (`ElementIndex`), but each element index (`ElementIndex`) can be associated with only one element type (`StreamElementType`).

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

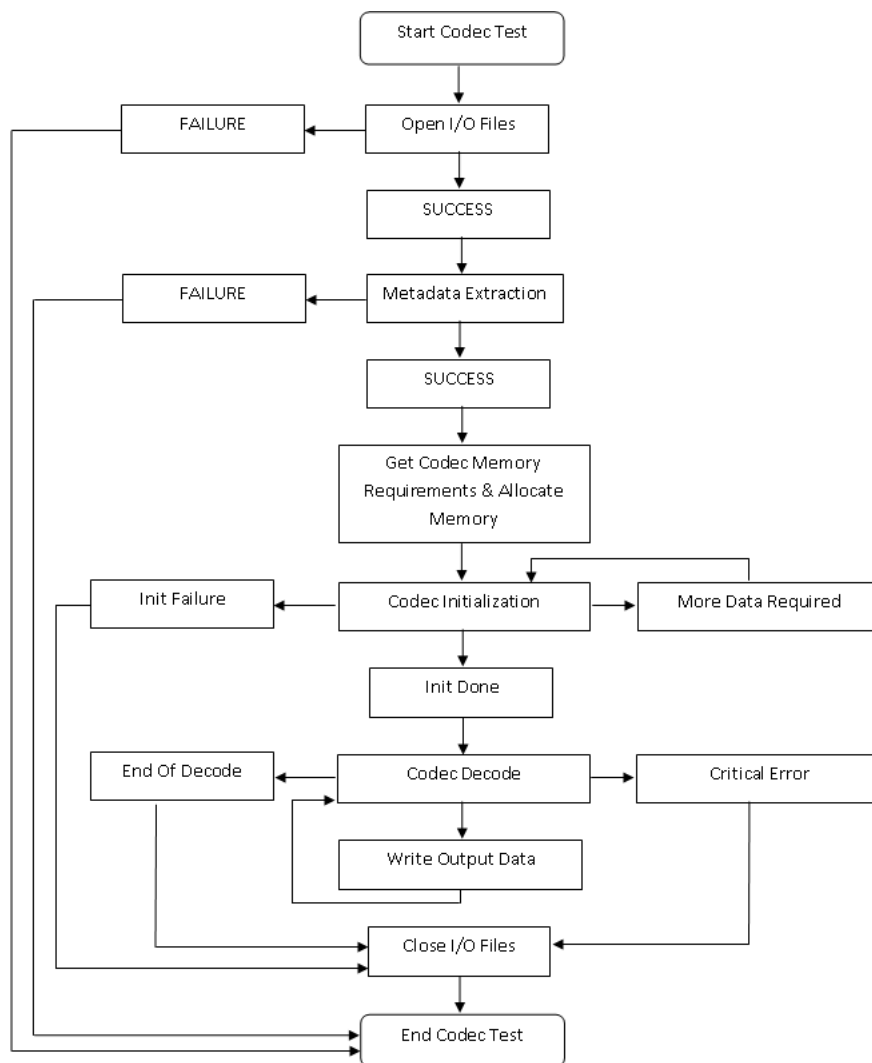
file_meta_data_t structure. The file_meta_data_t structure must be allocated by the application.

- 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_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

5. 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_CODEEC_WM8962 macro to 0 in the app_definitions.h file
5. Set the DEMO_CODEEC_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](#).

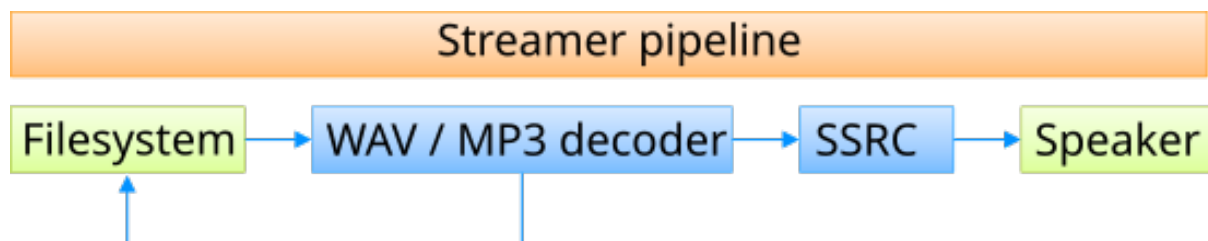
Processing Time

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.

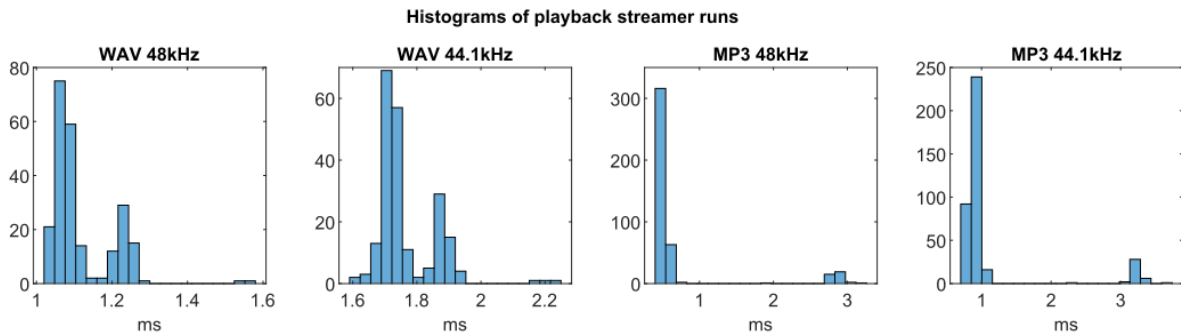
In the scenario involving the WAV codec, the audio file was accessed in every iteration of the streamer pipeline. Meaning, during each run, the file was read directly from the SD card. However, in the case of the MP3 codec, where data processing necessitates complete MP3 frames, the file wasn't read 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:

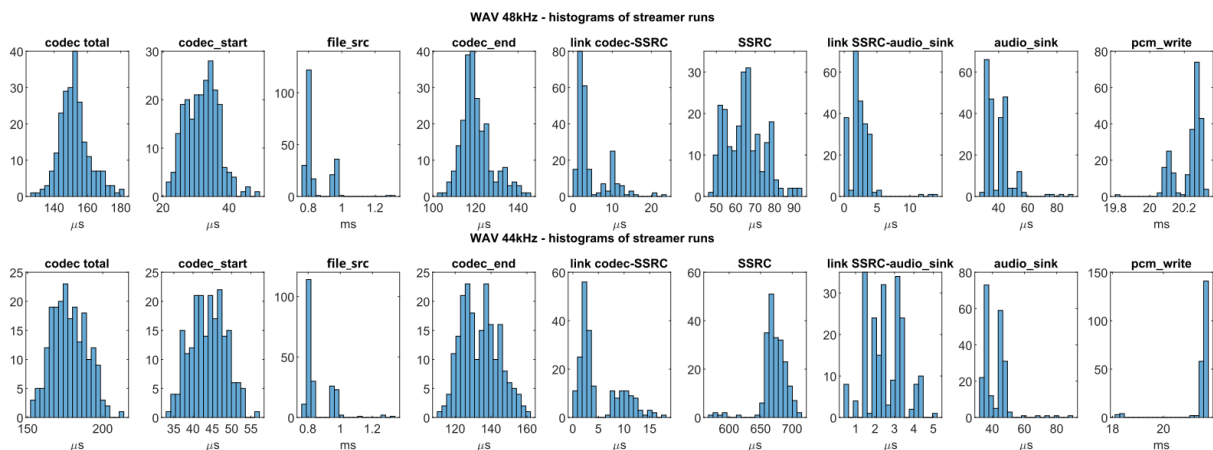
- **streamer** - total time of one pipeline run without time on output buffers
- **codec start** - time on decoder before file read
- **codec end** - time on decoder after file read
- **codec total** - codec_start+codec_end
- **file_src** - file reading time
- **SSRC_proc** - time on SSRC element
- **audio_sink** - 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

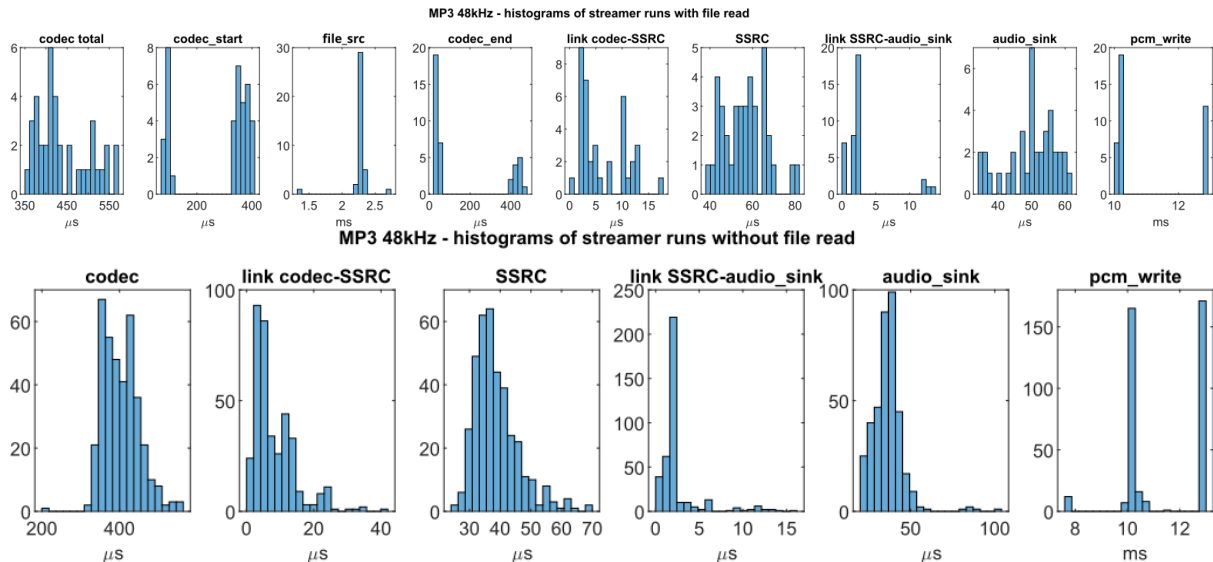
WAV 48kHz	stream	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_	link audio_sink	SSRC- audio_sink	au- dio_sin	pcm_write
mean	1.119 ms	152 µs	31 µs	0.843 ms	120 µs	5 µs	64 µs	2 µs		40 µs	20.228 ms
min	1.026 ms	125 µs	21 µs	0.773 ms	104 µs	<1 µs	47 µs	<1 µs		30 µs	19.805 ms
max	1.290 ms	193 µs	49 µs	1.311 ms	144 µs	23 µs	93 µs	14 µs		91 µs	20.324 ms

WAV 44kHz	stream	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_	link audio_sink	SSRC- audio_sink	au- dio_sin	pcm_write
mean	1.765 ms	178 µs	44 µs	0.853 ms	134 µs	5 µs	671 µs	3 µs		42 µs	21.472 ms
min	1.604 ms	145 µs	33 µs	0.770 ms	112 µs	<1 µs	574 µs	<1 µs		33 µs	18.163 ms
max	2.233 ms	218 µs	57 µs	1.335 ms	161 µs	18 µs	715 µs	5 µs		89 µs	21.746 ms



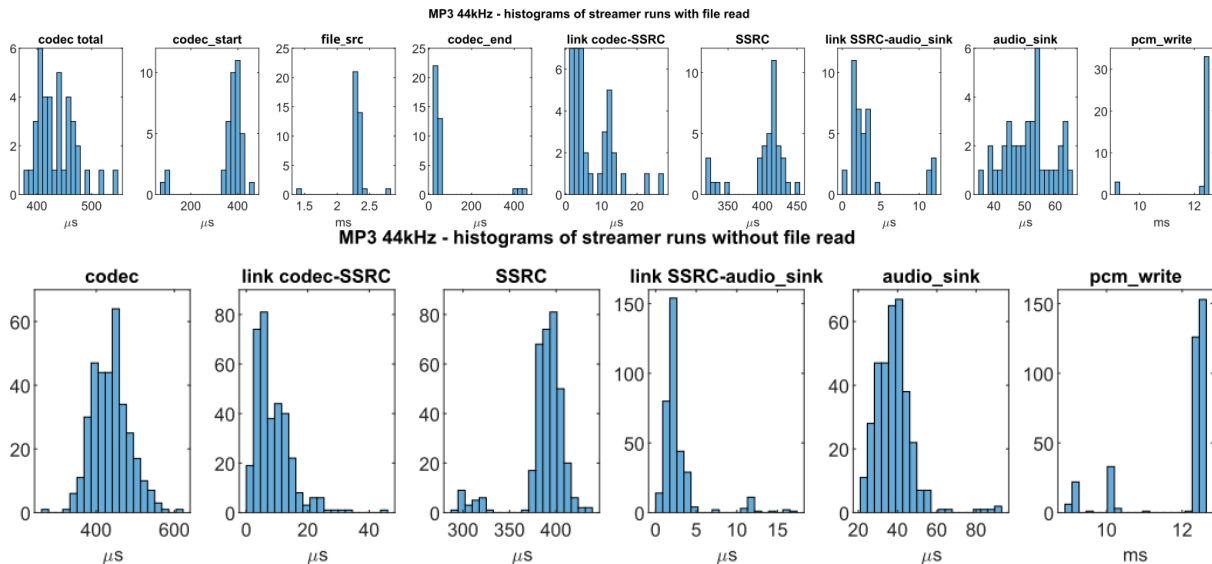
MP3 48 kHz w/ file read	stream	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_	link audio_sink	SSRC- audio_sink	au- dio_sin	pcm_write
mean	2.871 ms	441 µs	279 µs	2.271 ms	162 µs	6 µs	56 µs	3 µs		50 µs	11.019 ms
min	2.739 ms	353 µs	74 µs	1.353 ms	26 µs	<1 µs	40 µs	<1 µs		34 µs	10.091 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

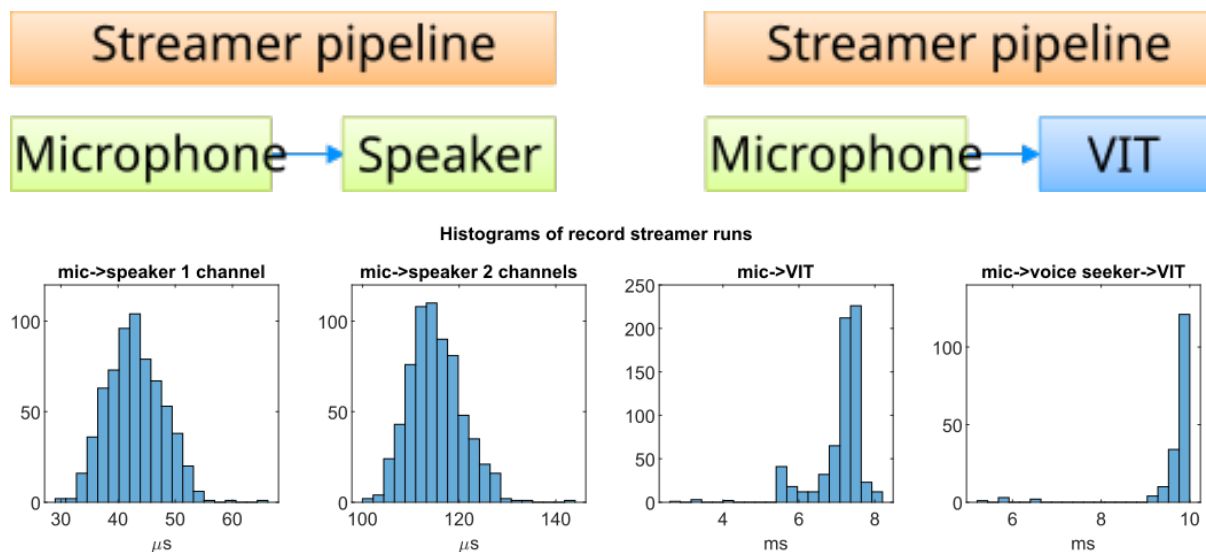


MP3 44 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_l	link SSRC- audio_sink	au- dio_sir	pcm_write
mean	3.217 ms	436 µs	367 µs	2.300 ms	66 µs	7 µs	403 µs	3 µs	51 µs	12.188 ms
min	2.329 ms	383 µs	73 µs	1.411 ms	26 µs	2 µs	318 µs	<1 µs	35 µs	9.119 ms
max	3.726 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

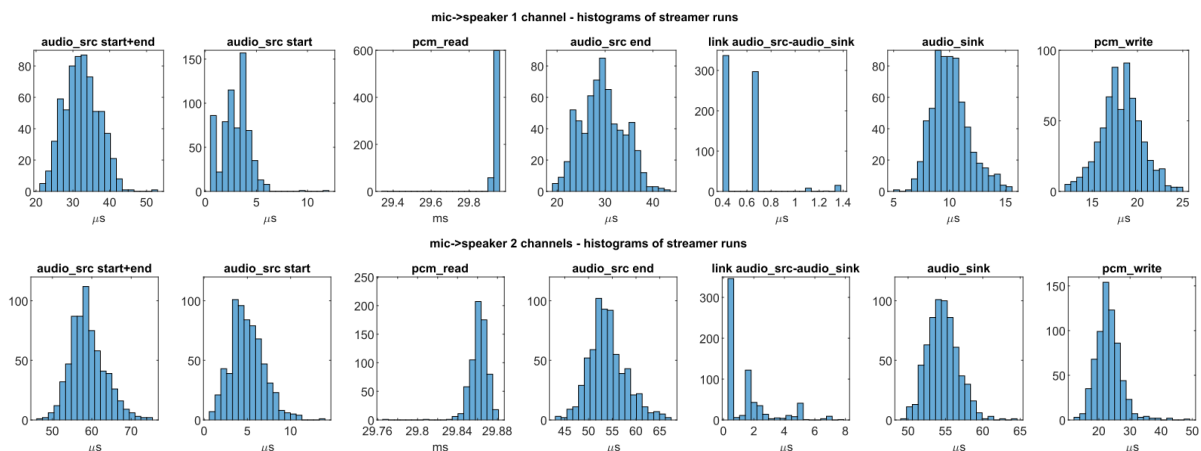
The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **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

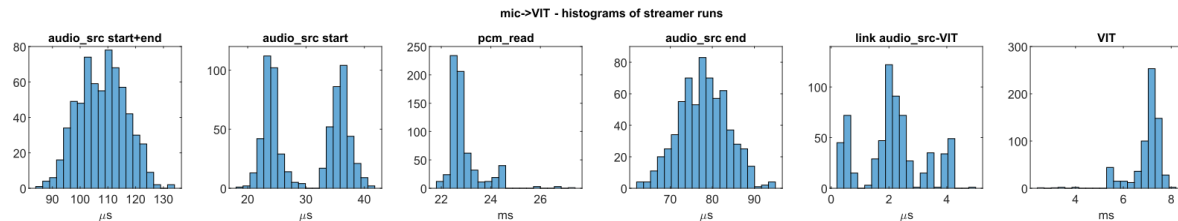
microphone speaker mono	->	stream	audio_src_start	pcm_read	audio_src_end	link audio_src-audio_sink	audio_sink	pcm_write
mean		43 μ s	3 μ s	29.938 ms	29 μ s	<1 μ s	10 μ s	18 μ s
min		26 μ s	<1 μ s	29.350 ms	19 μ s	<1 μ s	5 μ s	12 μ s
max		72 μ s	12 μ s	29.957 ms	44 μ s	1 μ s	15 μ s	25 μ s

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