



MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025



Table of contents

1	FRDM-MCXN236	3
1.1	Overview	3
1.2	Getting Started with MCUXpresso SDK Package	3
1.2.1	Getting Started with MCUXpresso SDK Package	3
1.3	Getting Started with MCUXpresso SDK GitHub	55
1.3.1	Getting Started with MCUXpresso SDK Repository	55
1.4	Release Notes	68
1.4.1	MCUXpresso SDK Release Notes	68
1.5	ChangeLog	73
1.5.1	MCUXpresso SDK Changelog	73
1.6	Driver API Reference Manual	177
1.7	Middleware Documentation	177
1.7.1	Multicore	178
1.7.2	Audio Voice components	178
1.7.3	Maestro Audio Framework for MCU	178
1.7.4	FreeMASTER	178
1.7.5	FreeRTOS	178
1.7.6	File systemFatfs	178
2	MCXN236	179
2.1	CACHE: LPCAC CACHE Memory Controller	179
2.2	CDOG	179
2.3	Clock Driver	183
2.4	CRC: Cyclic Redundancy Check Driver	231
2.5	CTIMER: Standard counter/timers	234
2.6	eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	243
2.7	eDMA core Driver	274
2.8	eDMA soc Driver	281
2.9	EIM: error injection module	281
2.10	ERM: error recording module	282
2.11	EVTG: Event Generator Driver	284
2.12	EVTG Driver	289
2.13	EWM: External Watchdog Monitor Driver	290
2.14	FGPIO Driver	293
2.15	FlexCAN: Flex Controller Area Network Driver	293
2.16	FlexCAN Driver	293
2.17	FlexCAN eDMA Driver	340
2.18	FlexIO: FlexIO Driver	343
2.19	FlexIO Driver	343
2.20	FlexIO eDMA I2S Driver	360
2.21	FlexIO eDMA SPI Driver	364
2.22	FlexIO eDMA UART Driver	367
2.23	FlexIO I2C Master Driver	370
2.24	FlexIO I2S Driver	378
2.25	FlexIO SPI Driver	389
2.26	FlexIO UART Driver	402

2.27	FREQME: Frequency Measurement	412
2.28	GDET	412
2.29	Glitch Detect	414
2.30	GPIO: General-Purpose Input/Output Driver	414
2.31	GPIO Driver	417
2.32	I3C: I3C Driver	422
2.33	I3C Common Driver	425
2.34	I3C Master Driver	427
2.35	I3C Master DMA Driver	453
2.36	I3C Slave Driver	456
2.37	I3C Slave DMA Driver	468
2.38	INPUTMUX: Input Multiplexing Driver	471
2.39	INTM: Interrupt Monitor Driver	579
2.40	IRTC: IRTC Driver	582
2.41	Intrusion and Tamper Response Controller	591
2.42	ITRC	591
2.43	Common Driver	594
2.44	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	606
2.45	Lpc_freqme	627
2.46	LPCMP: Low Power Analog Comparator Driver	632
2.47	LPFLEXCOMM: LPFLEXCOMM Driver	642
2.48	LPFLEXCOMM Driver	642
2.49	LPI2C: Low Power Inter-Integrated Circuit Driver	643
2.50	LPI2C Master Driver	644
2.51	LPI2C Master DMA Driver	657
2.52	LPI2C Slave Driver	660
2.53	LPSPi: Low Power Serial Peripheral Interface	670
2.54	LPSPi Peripheral driver	670
2.55	LPSPi eDMA Driver	692
2.56	LPTMR: Low-Power Timer	697
2.57	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	703
2.58	LPUART Driver	703
2.59	LPUART eDMA Driver	723
2.60	MCX_CMC: Core Mode Controller Driver	726
2.61	MCX_SPC: System Power Control driver	738
2.62	MCX_VBAT: Smart Power Switch	778
2.63	MRT: Multi-Rate Timer	798
2.64	OSTIMER: OS Event Timer Driver	802
2.65	OTP: One-Time Programmable memory and API	806
2.66	PDM: Microphone Interface	809
2.67	PDM Driver	809
2.68	PDM EDMA Driver	821
2.69	PINT: Pin Interrupt and Pattern Match Driver	825
2.70	PORT: Port Control and Interrupts	833
2.71	PUF: Physical Unclonable Function	841
2.72	Puf_v3_driver	844
2.73	PWM: Pulse Width Modulator	849
2.74	QDC: Quadrature Encoder/Decoder	874
2.75	Reset Driver	885
2.76	RTC: Real Time Clock	889
2.77	SAI: Serial Audio Interface	894
2.78	SAI Driver	894
2.79	SAI EDMA Driver	920
2.80	SMARTDMA: SMART DMA Driver	927
2.81	SYSPM: System Performance Monitor	928
2.82	Digital Tamper	931
2.83	TDET	931
2.84	TRDC: Trusted Resource Domain Controller	945

2.85	Trdc_core	968
2.86	UTICK: MicroTick Timer Driver	982
2.87	VREF: Voltage Reference Driver	983
2.88	WUU: Wakeup Unit driver	985
2.89	WWDT: Windowed Watchdog Timer Driver	990
3	Middleware	995
3.1	Motor Control	995
3.1.1	FreeMASTER	995
3.2	Multimedia	1032
3.2.1	Audio Voice	1032
4	RTOS	1087
4.1	FreeRTOS	1087
4.1.1	FreeRTOS kernel	1087
4.1.2	FreeRTOS drivers	1087
4.1.3	backoffalgorithm	1087
4.1.4	corehttp	1087
4.1.5	corejson	1087
4.1.6	coremqtt	1088
4.1.7	corepkcs11	1088
4.1.8	freertos-plus-tcp	1088

This documentation contains information specific to the frdm-mcxn236 board.

Chapter 1

FRDM-MCXN236

1.1 Overview

The NXP FRDM-MCXN236 is a development board for the N23x 150 MHz Arm Cortex-M33 TrustZone microcontroller, which is for Industrial and Consumer IoT Applications.



MCU device and part on board is shown below:

- Device: MCXN236
- PartNumber: MCXN236VDF

1.2 Getting Started with MCUXpresso SDK Package

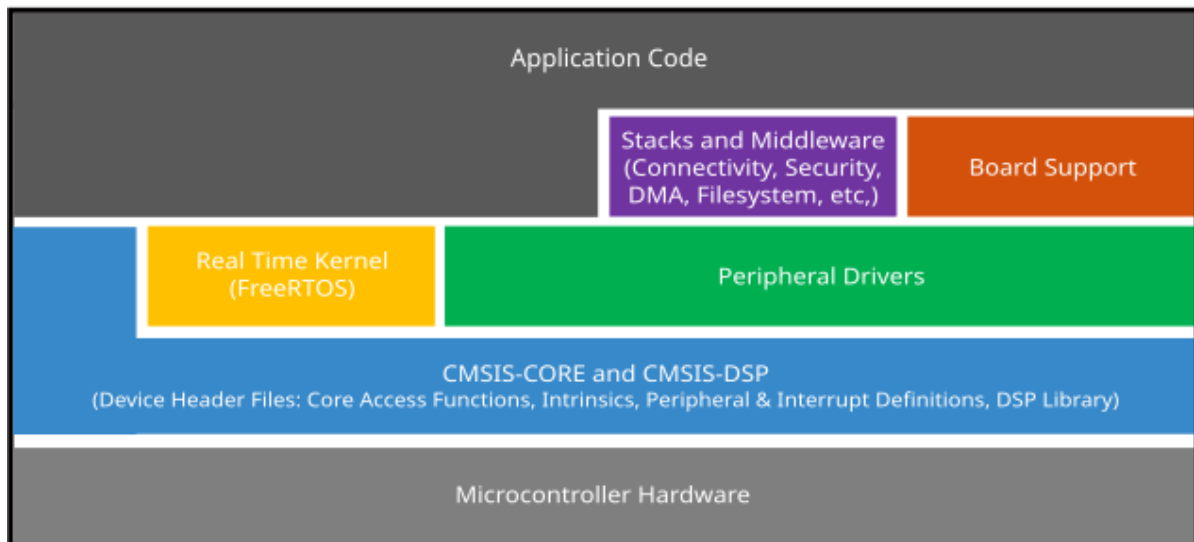
1.2.1 Getting Started with MCUXpresso SDK Package

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, 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

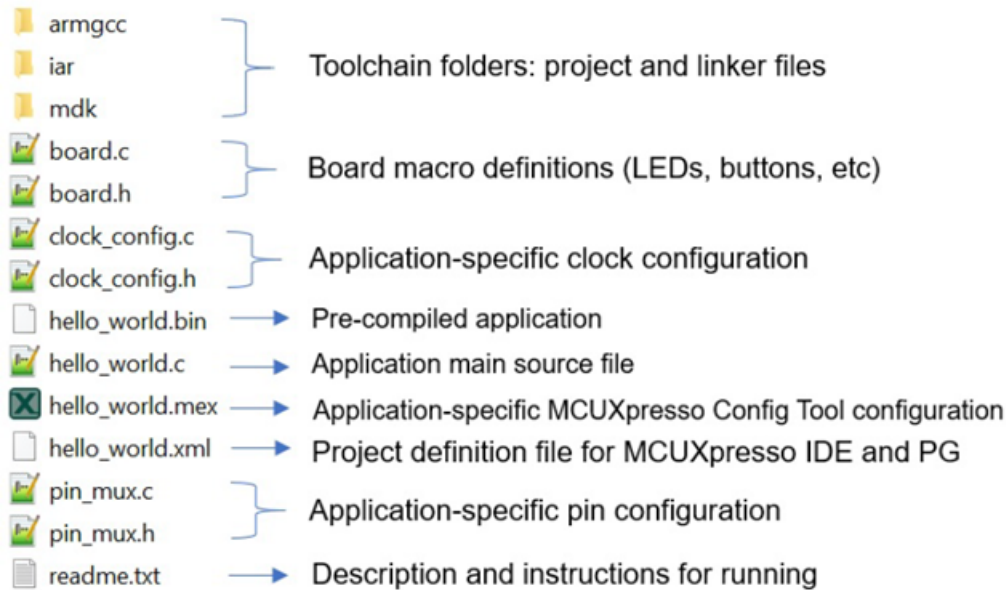
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each `<board_name>` folder, there are various subfolders to classify the type of examples it contains. 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.

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

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

- `devices/<device_name>`: The device's CMSIS header file, MCUXpresso SDK feature file, and a few other files
- `devices/<device_name>/cmsis_drivers`: All the CMSIS drivers for your specific MCU
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code, including vector table definitions
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications
- `devices/<device_name>/project`: Project template used in CMSIS PACK new project creation

For examples containing 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.

Run a demo using MCUXpresso IDE

Note: Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

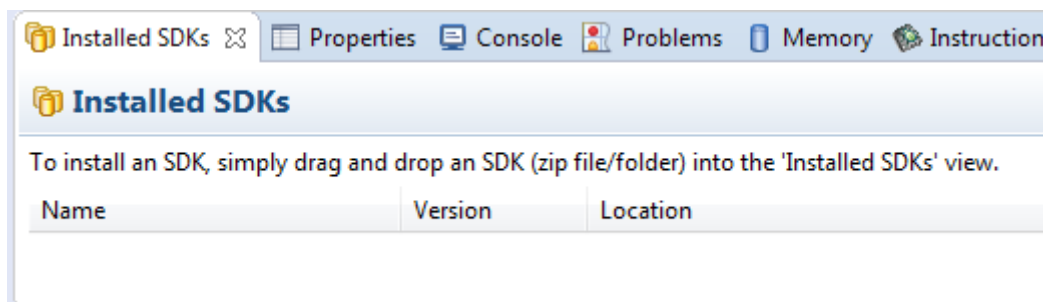
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The `hello_world` demo application targeted for the hardware platform is

used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

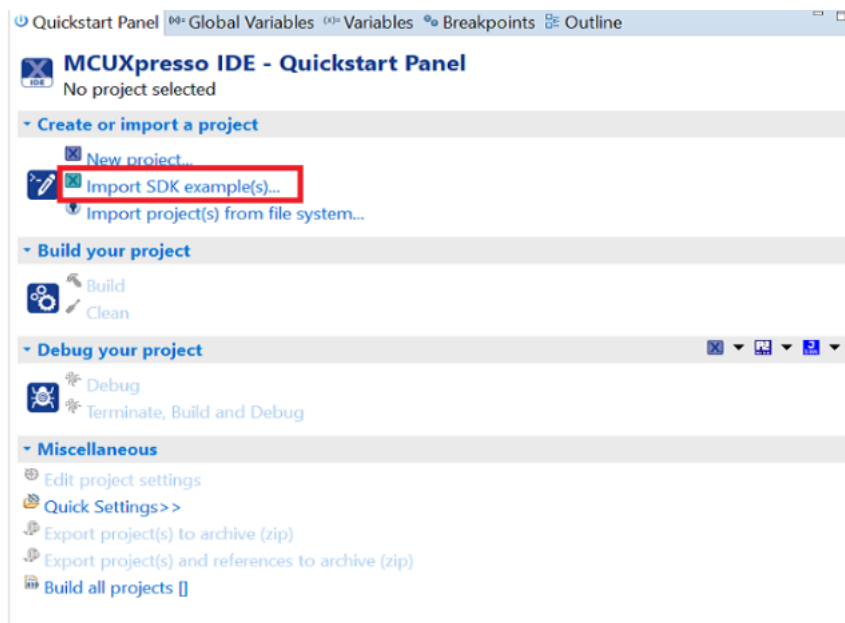
Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace.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

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

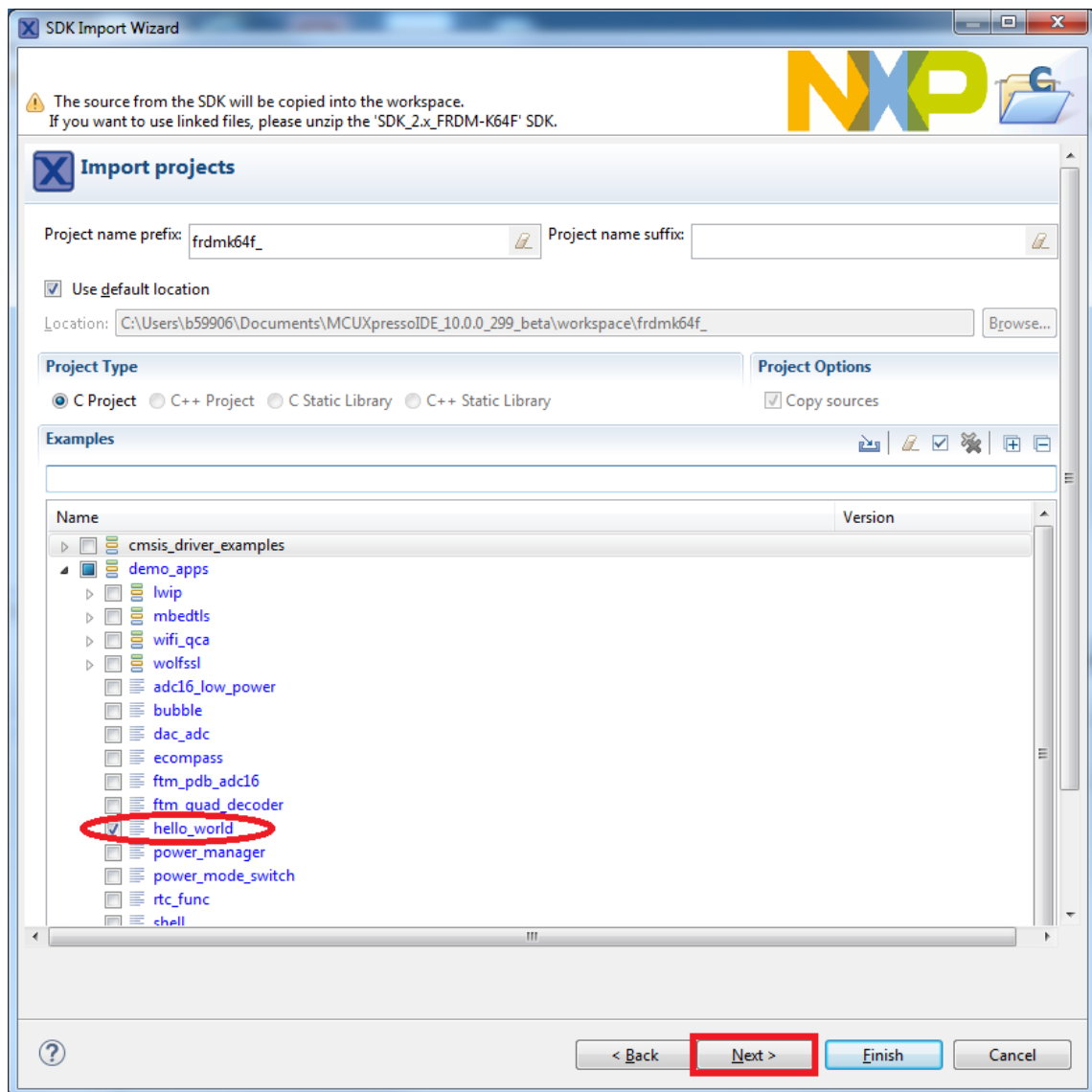
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



3. Expand the `demo_apps` folder and select `hello_world`.
4. Click **Next**.



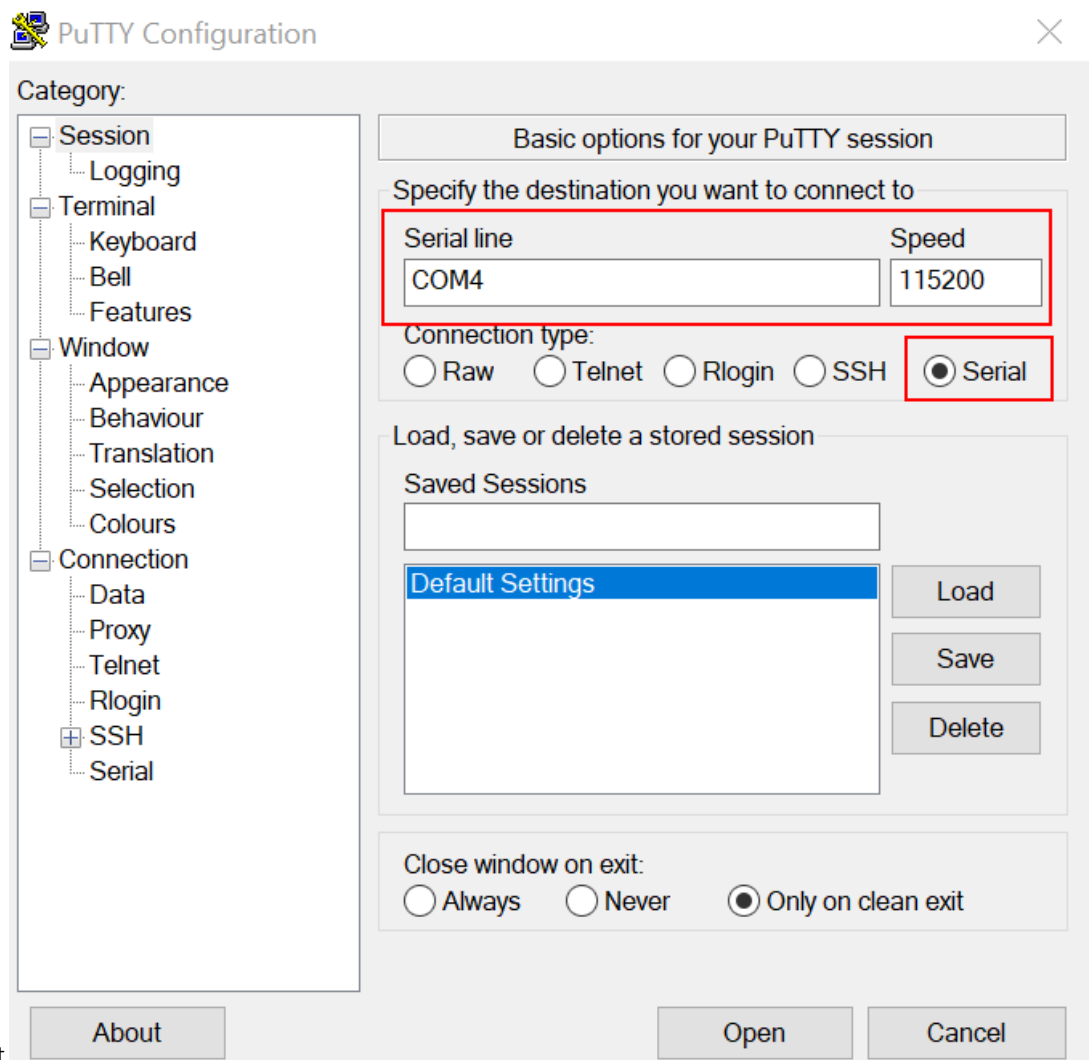
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

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:

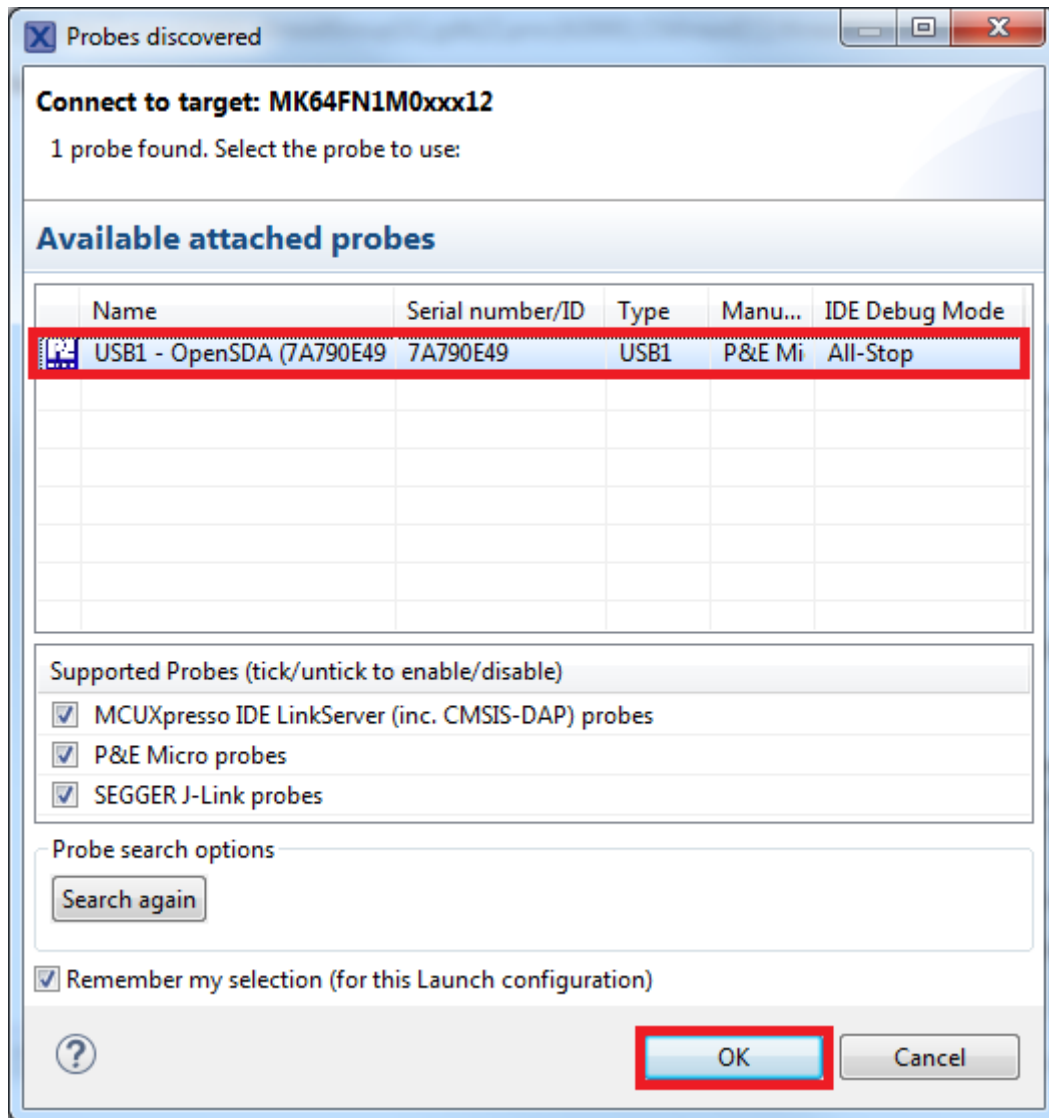
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM 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 **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()`.
- Start the application by clicking **Resume**.

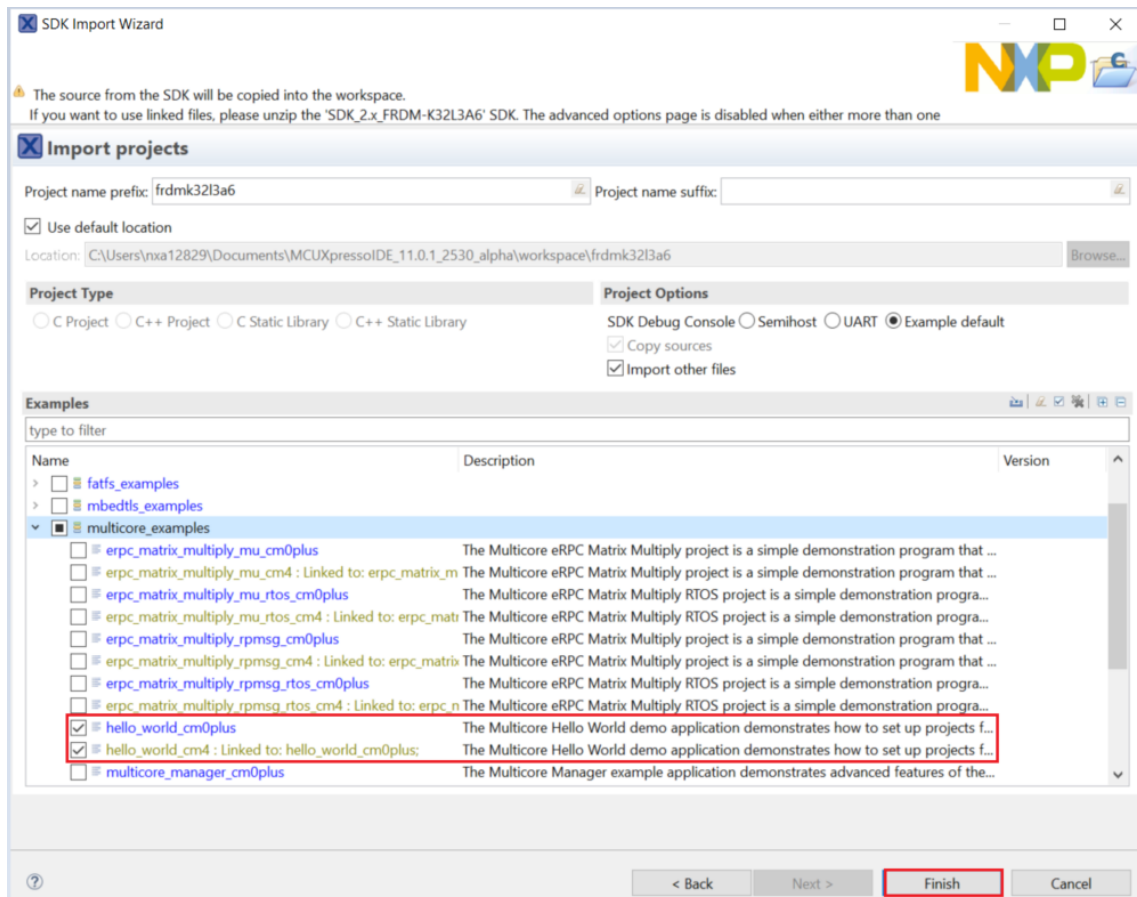


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

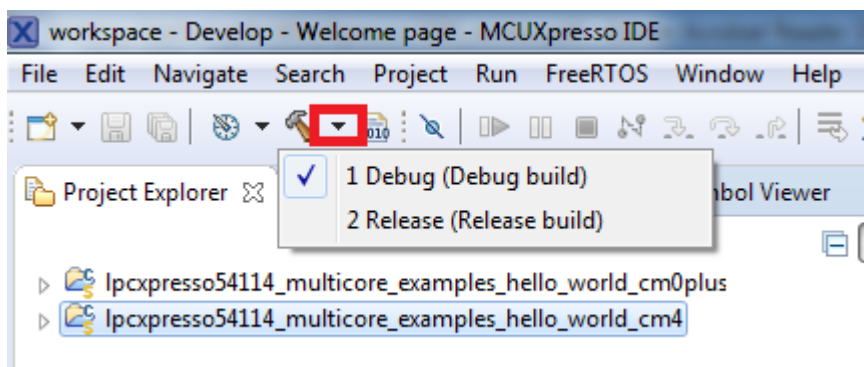


Build a multicore example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world example application targeted for the LPCXpresso54114 hardware platform is used as an example.

1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in **Build an example application**. When the SDK zip package for LPCXpresso54114 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 **LPCxx** folder and select **LPC54114J256**. Then, select **lpcxpresso54114** and click **Next**.
2. Expand the multicore_examples/hello_world folder and select **cm4**. The cm0plus counterpart project is automatically imported with the cm4 project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.

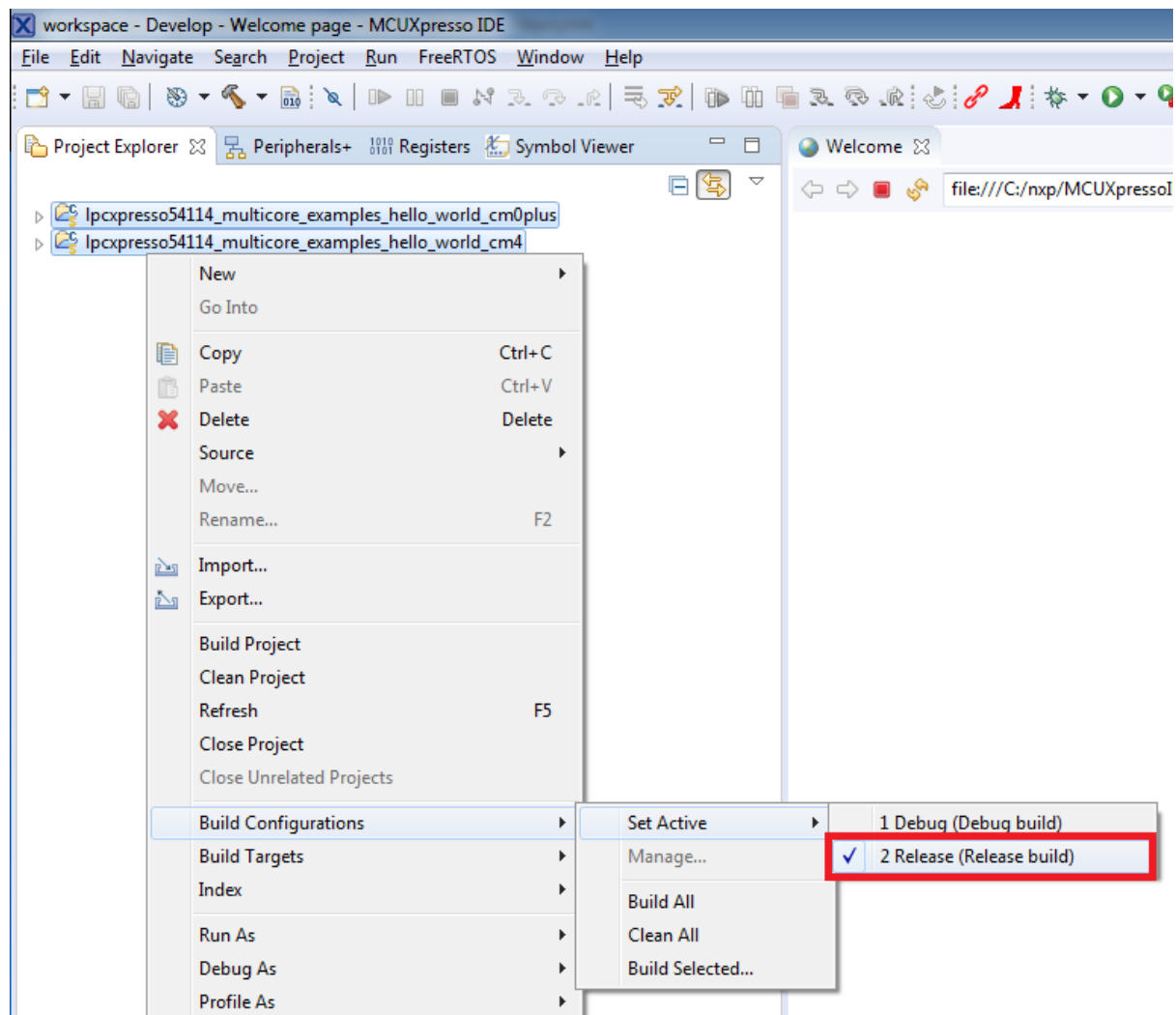


3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by clicking the downward facing arrow next to the hammer icon, as shown in the figure. For this example, select **Debug**.

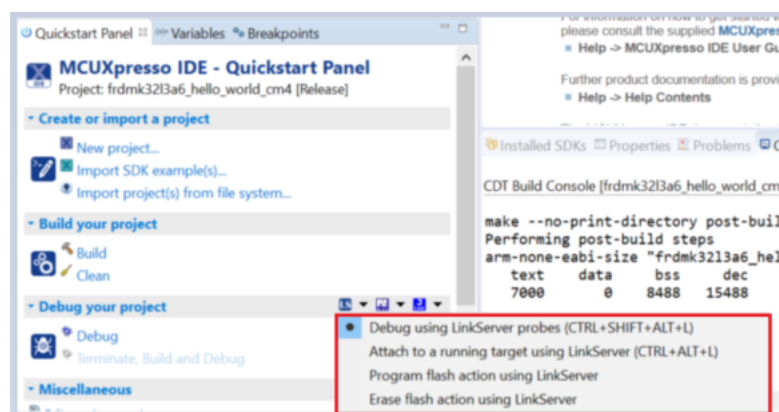


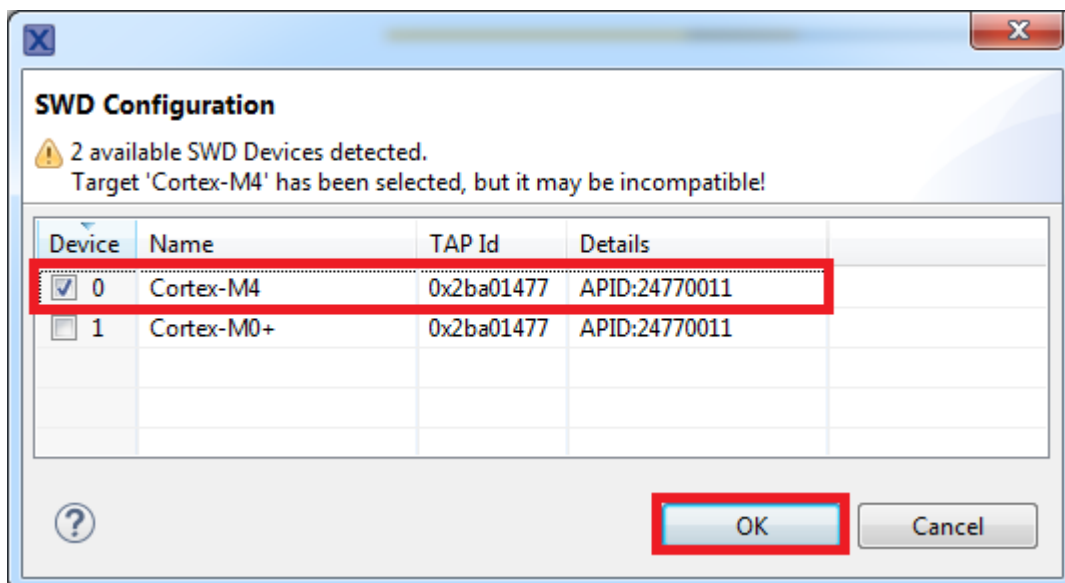
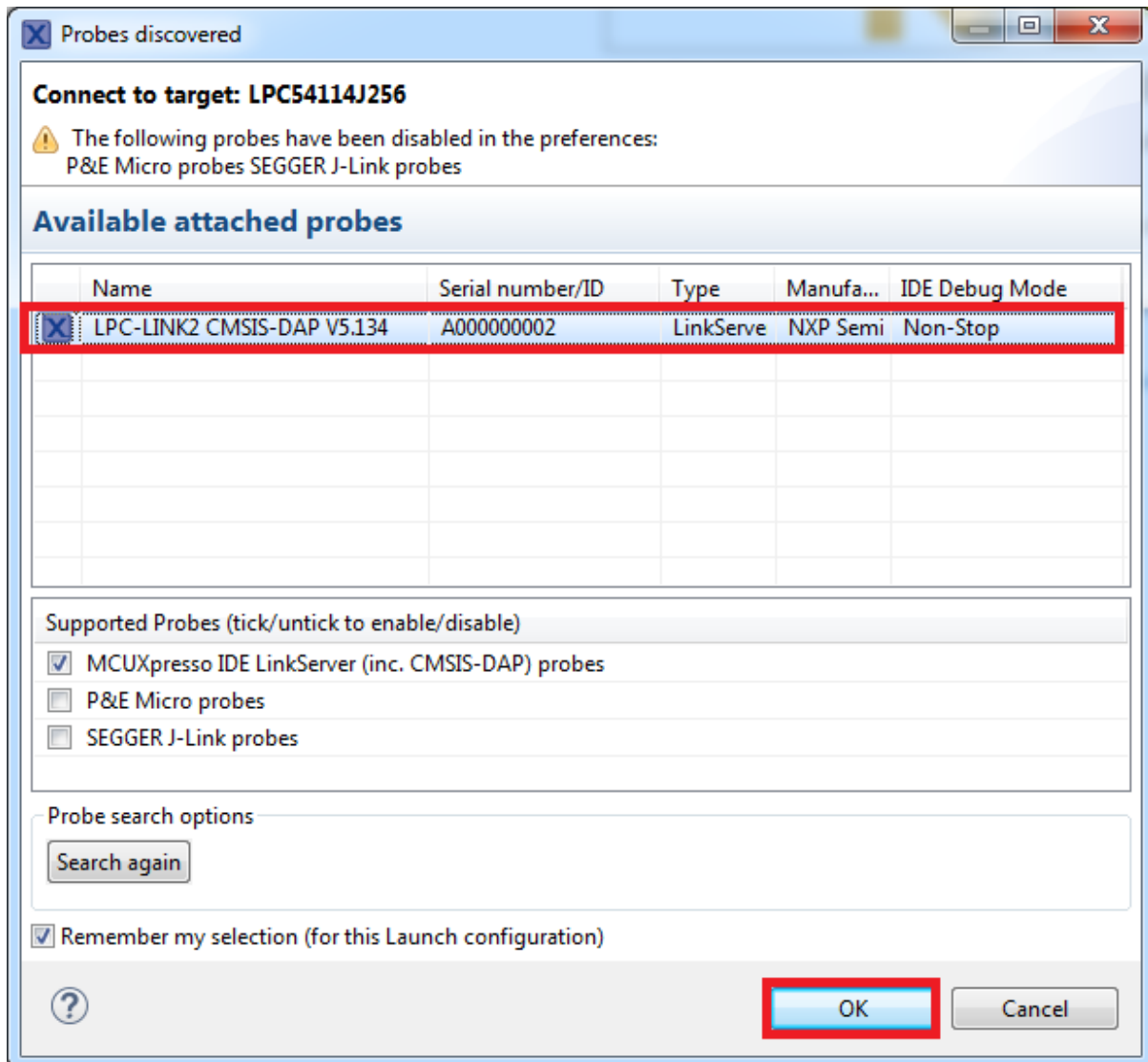
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

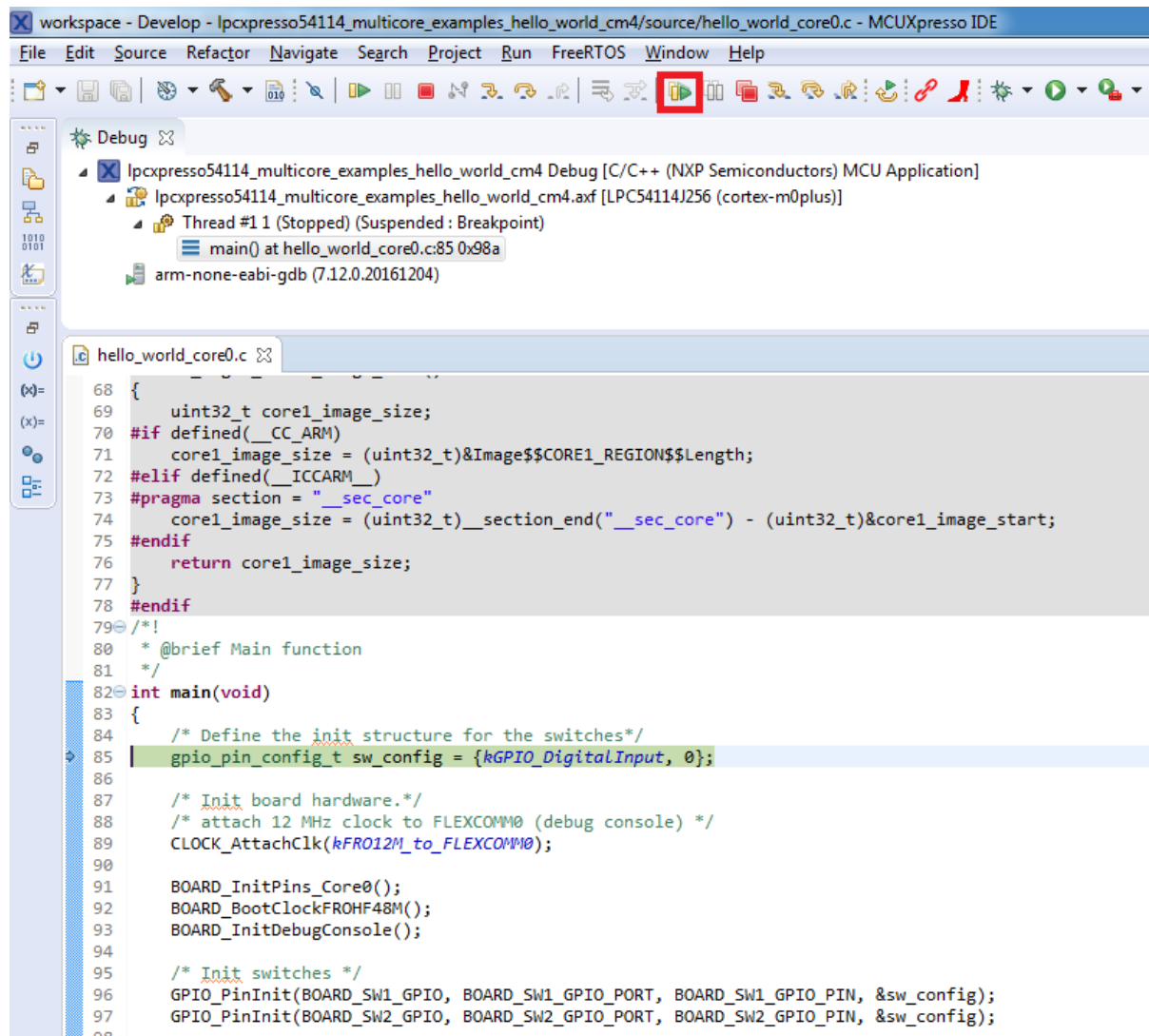
Note: When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build Configurations -> Set Active -> Release**. This alternate navigation using the menu item is **Project -> Build Configuration -> Set Active -> Release**. After switching to the **Release** build configuration, the build of the multicore example can be started by triggering the primary core application (cm4) build.



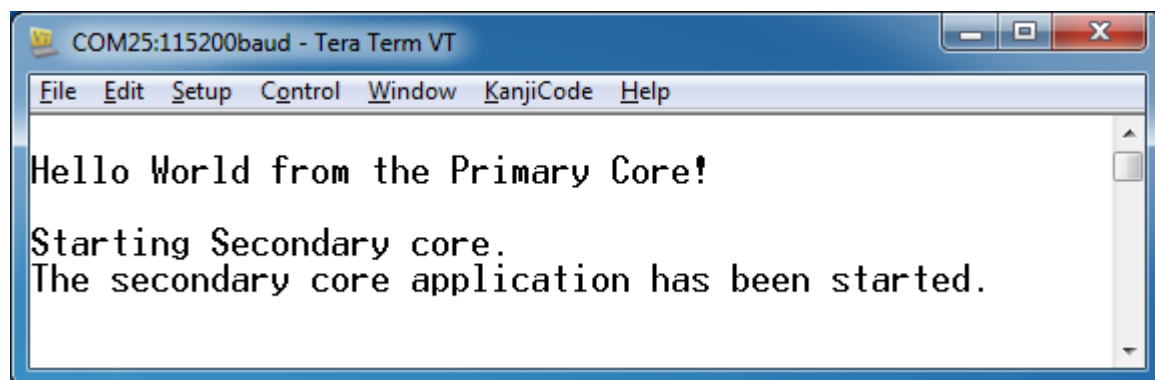
Run a multicore example application The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run an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





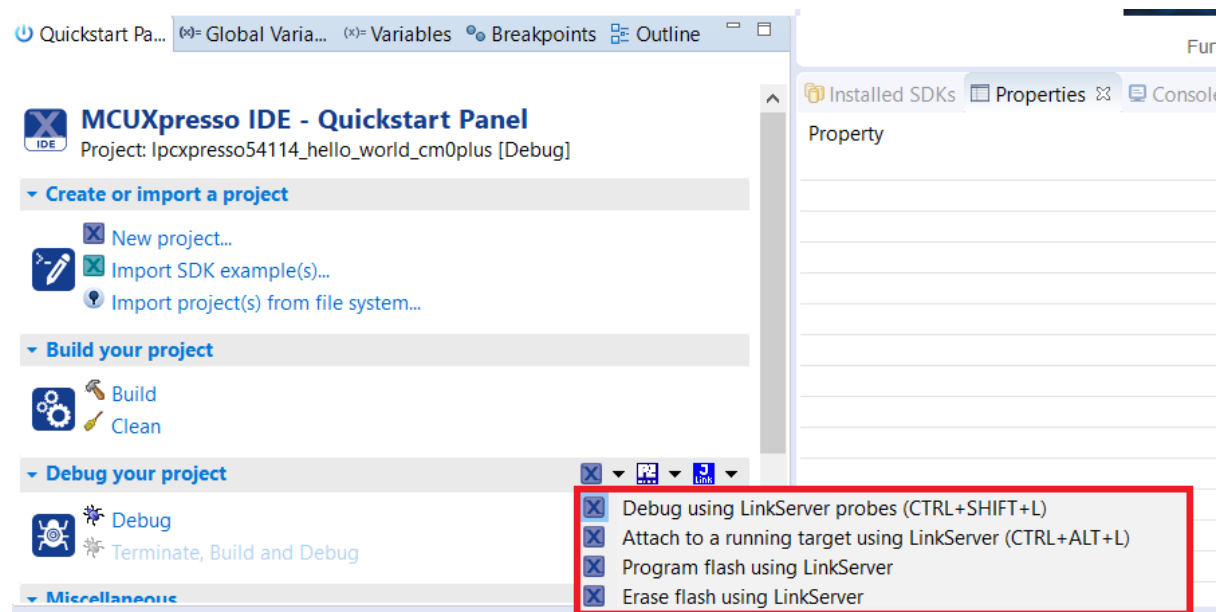


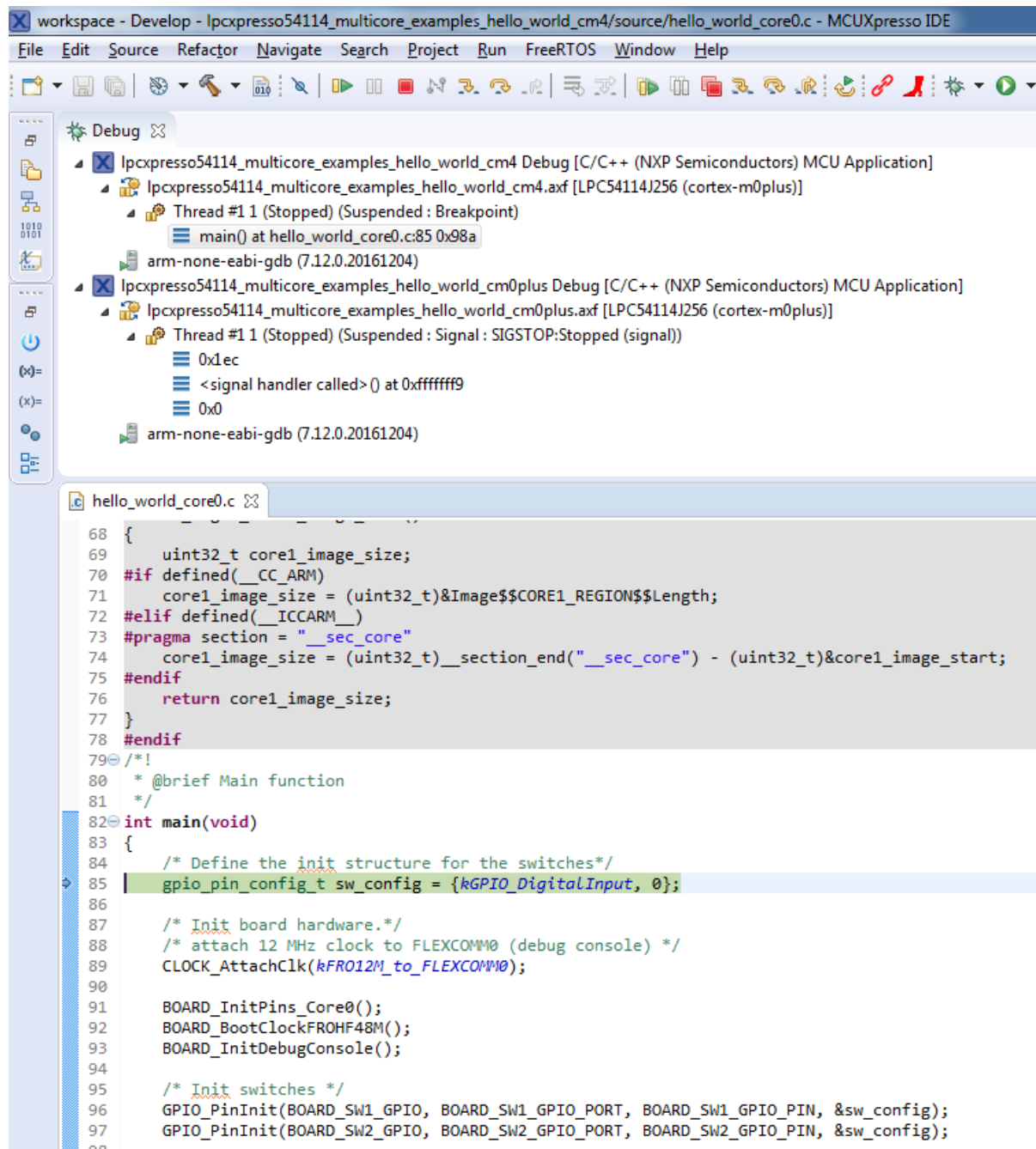
After clicking the “Resume All Debug sessions” button, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



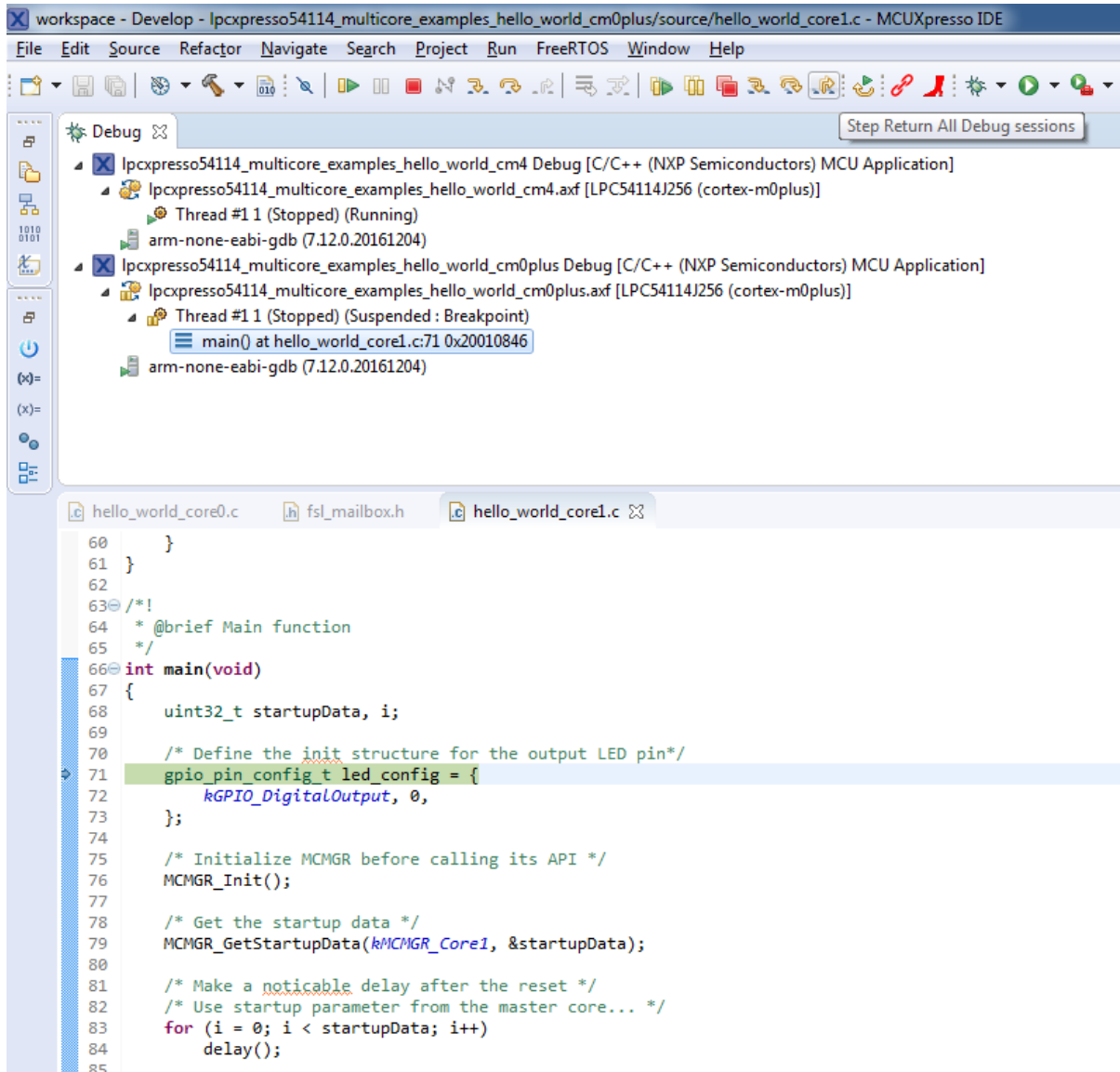
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps also for the auxiliary core application. Highlight the lpcxpresso54114_multicore_examples_hello_world_cm0plus project (multicore slave project) in the Project Explorer. On the Quickstart Panel, click “Debug ‘lpcxpresso54114_multicore_examples_hello_world_cm0plus’ [Debug]” to launch the second debug

session.

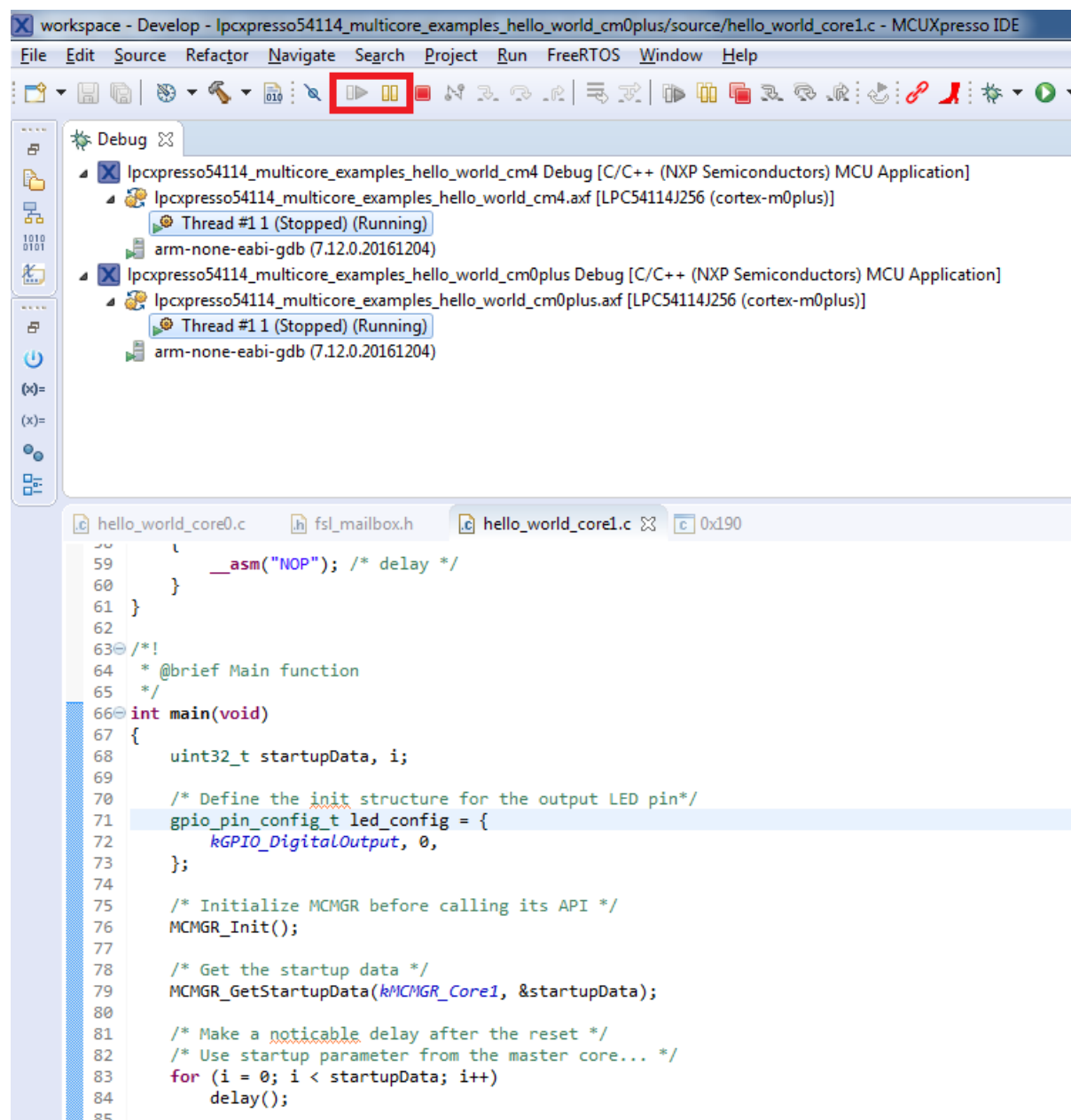


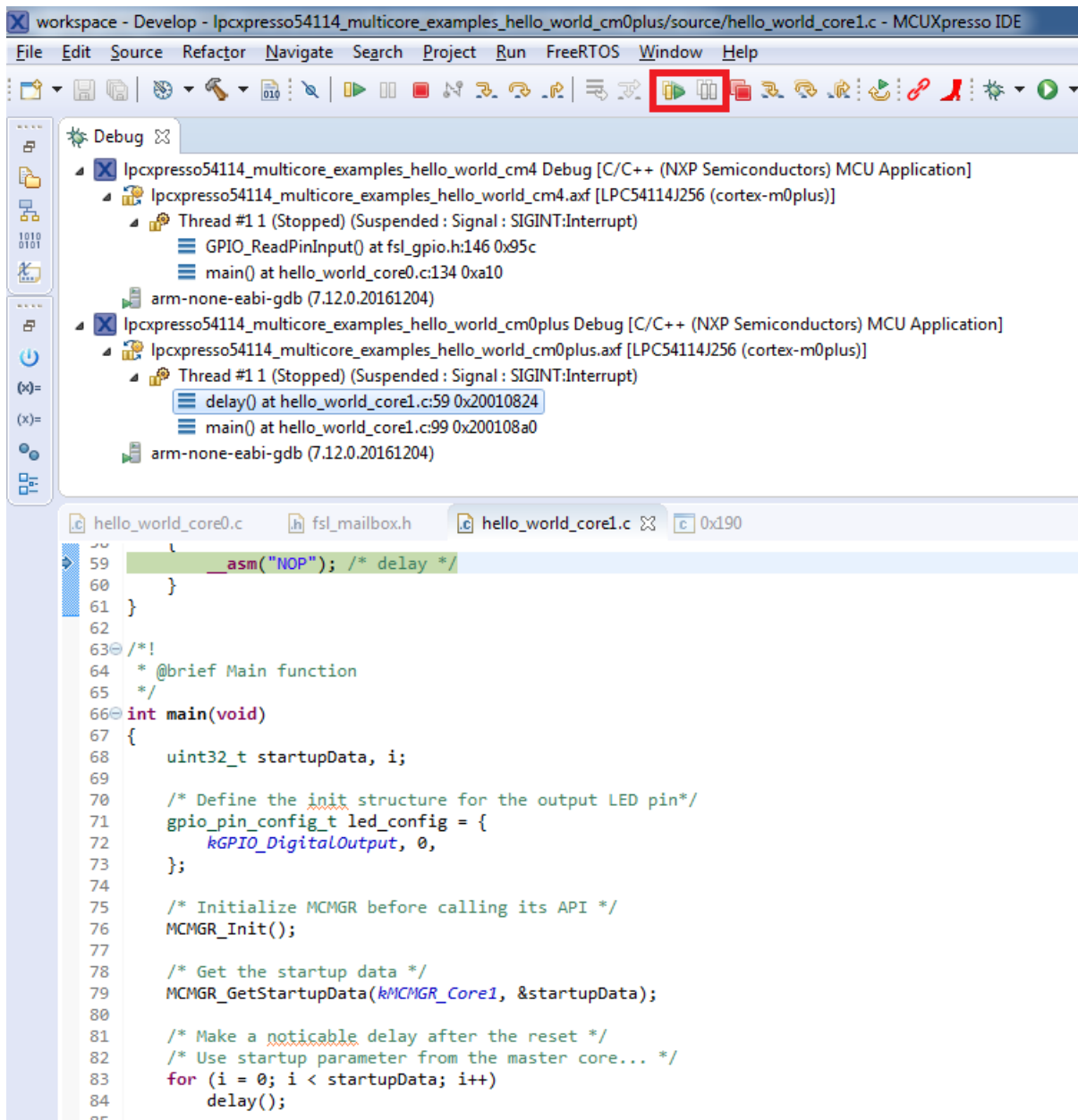


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello_world multicore application then starts running. The primary core application starts the auxiliary core application during runtime, and the auxiliary core application stops at the beginning of the main() function. The debug session of the auxiliary core application is highlighted. After clicking the “Resume” button, it is applied to the auxiliary core debug session. Therefore, the auxiliary core application continues its execution.



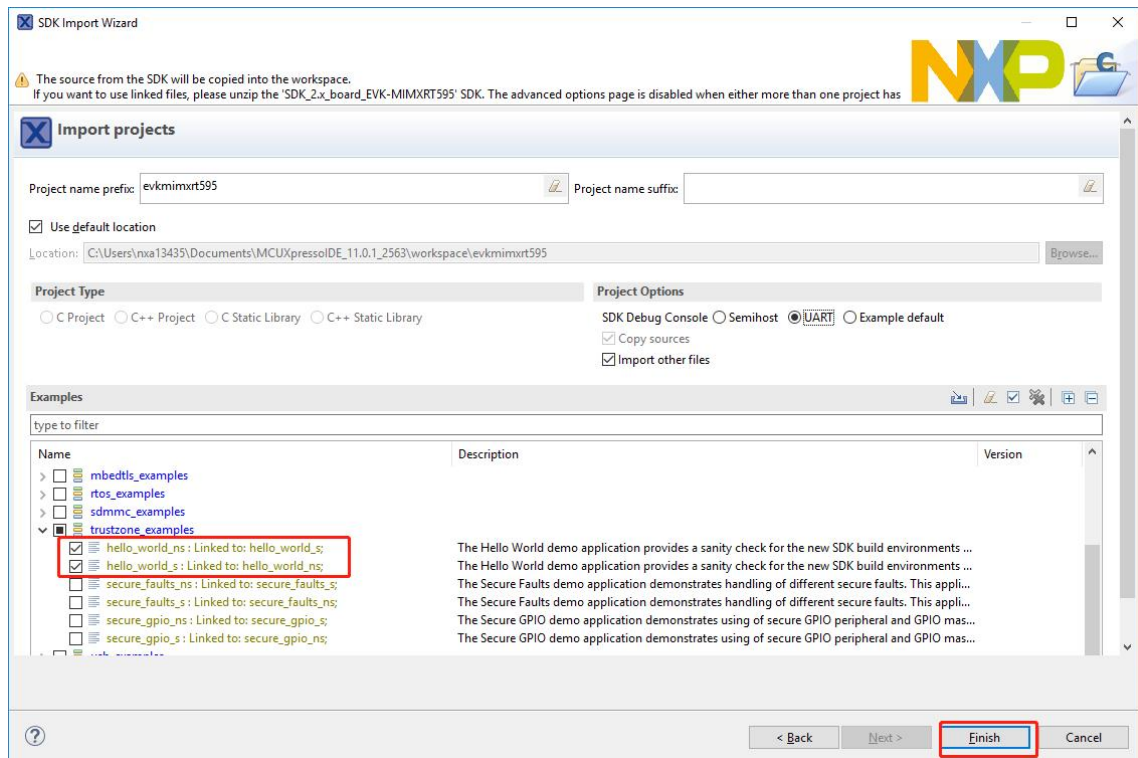
At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both the cores. This is done either by selecting both opened debug sessions (multiple selections) and clicking the “Suspend” / “Resume” control button, or just using the “Suspend All Debug sessions” and the “Resume All Debug sessions” buttons.



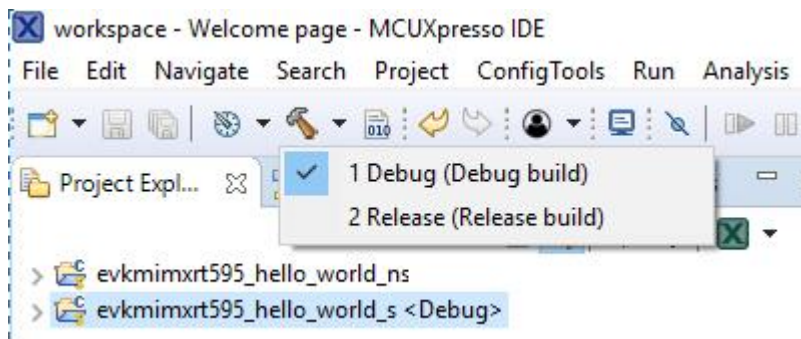


Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the hello_world example application targeted for the MIMXRT595-EVK hardware platform is used as an example, though 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 MIMXRT595-EVK 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 **MIMXRT500** folder and select **MIMXRT595S**. Then, select **evkmimxrt595** 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, click **Finish**.

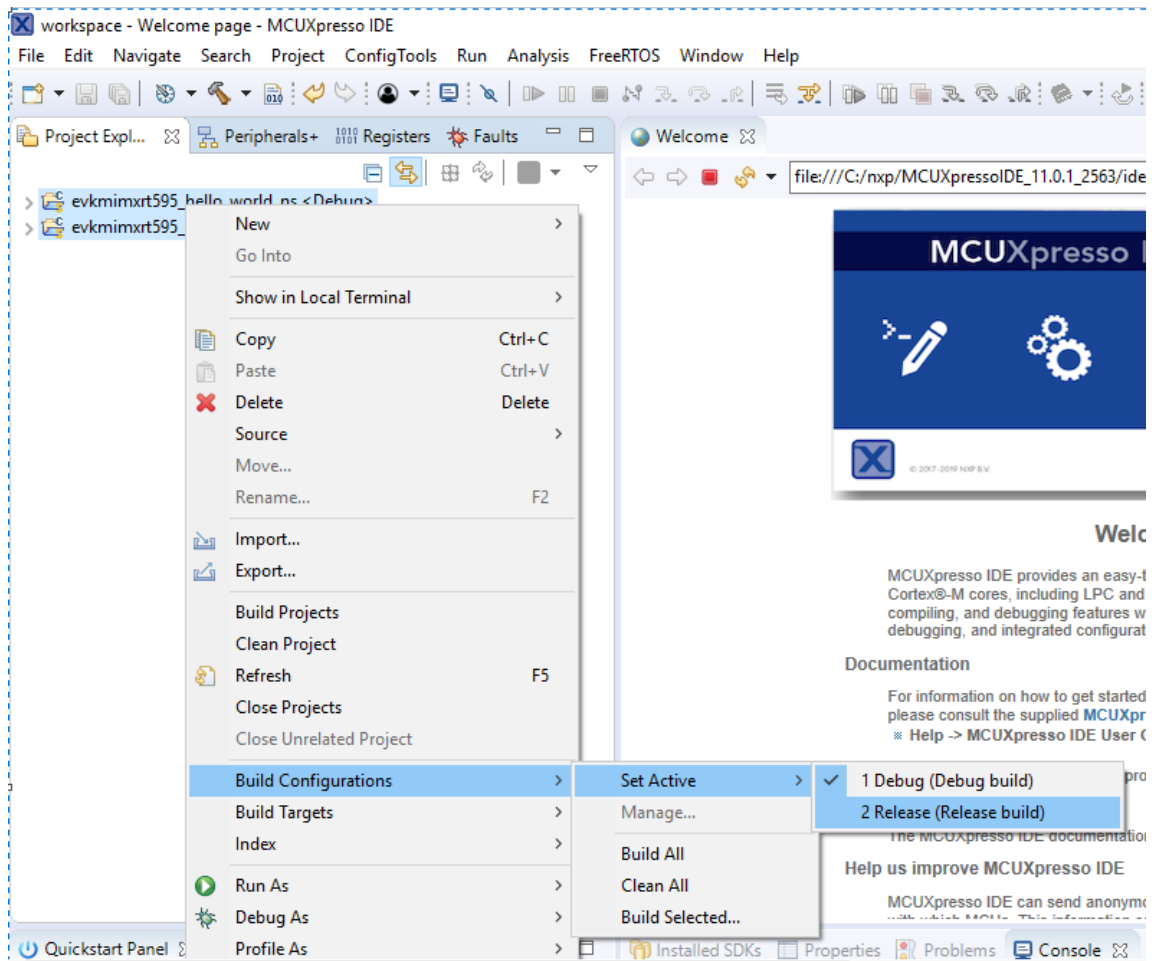


- Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward facing arrow next to the hammer icon, as shown in following figure. For this example, select the **Debug** target.



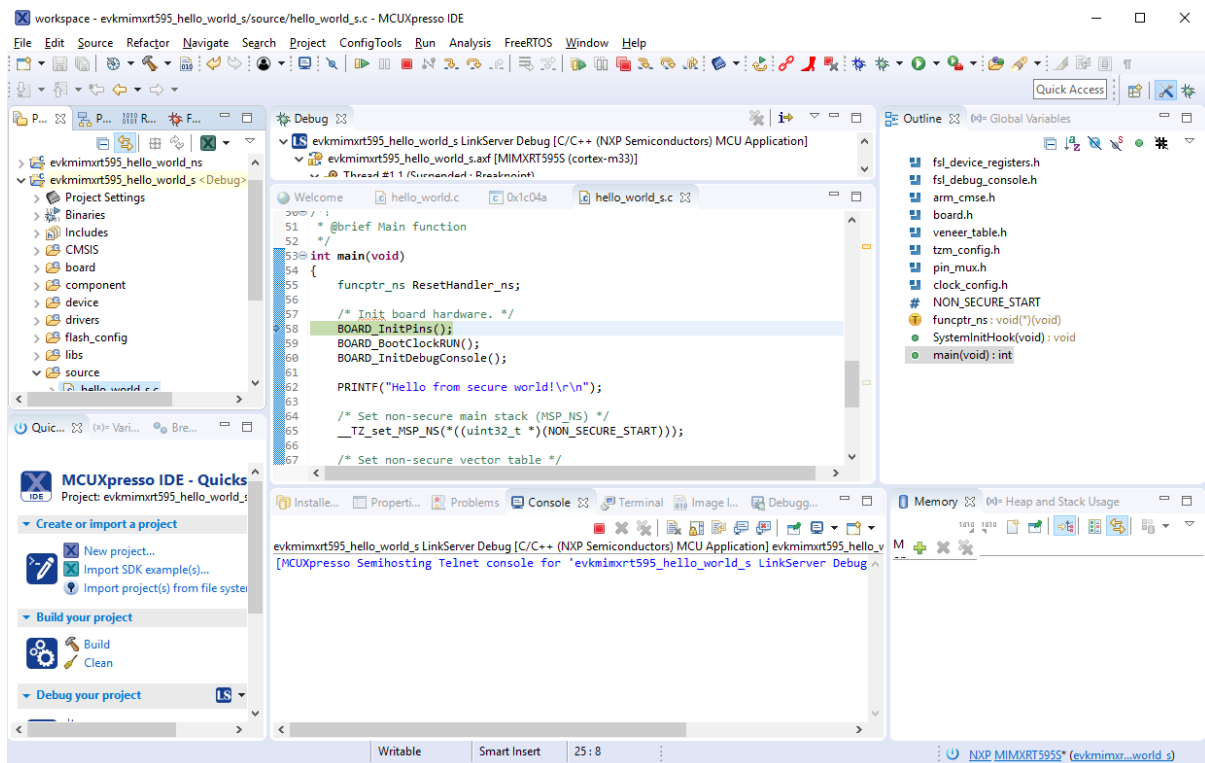
The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when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



Run a TrustZone example application To download and run the application, perform all steps as described in **Run an example application**. These steps are common for single core, and TrustZone applications, ensuring <board_name>_hello_world_s is selected for debugging.

In the Quickstart Panel, click **Debug** to launch the second debug session.



Now, the TrustZone sessions should be opened. Click **Resume**. The `hello_world` TrustZone application then starts running, and the secure application starts the non-secure application during runtime.

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.

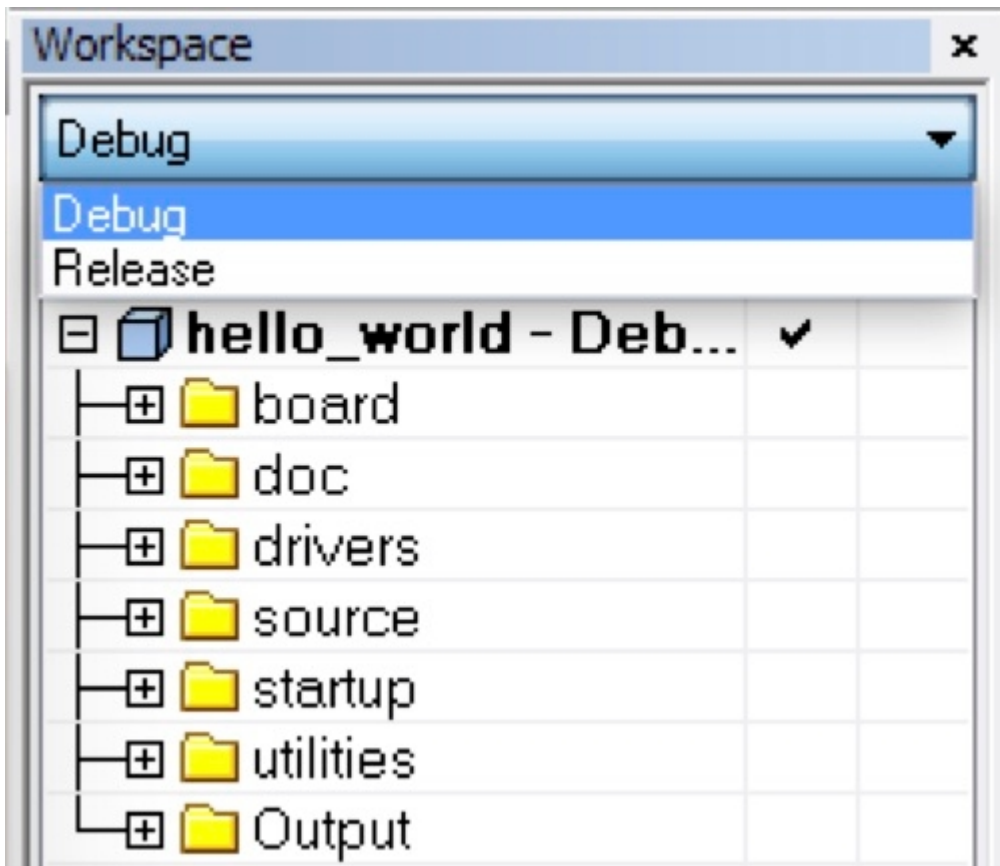
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

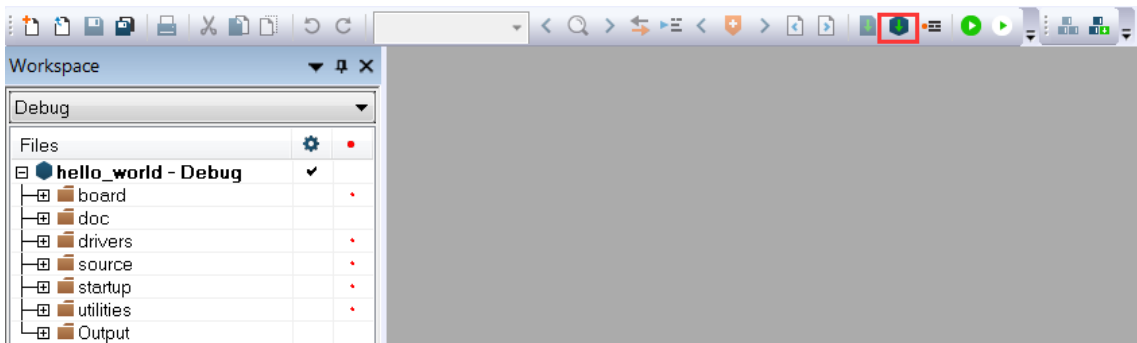
Other example applications may have additional folders in their path.

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

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



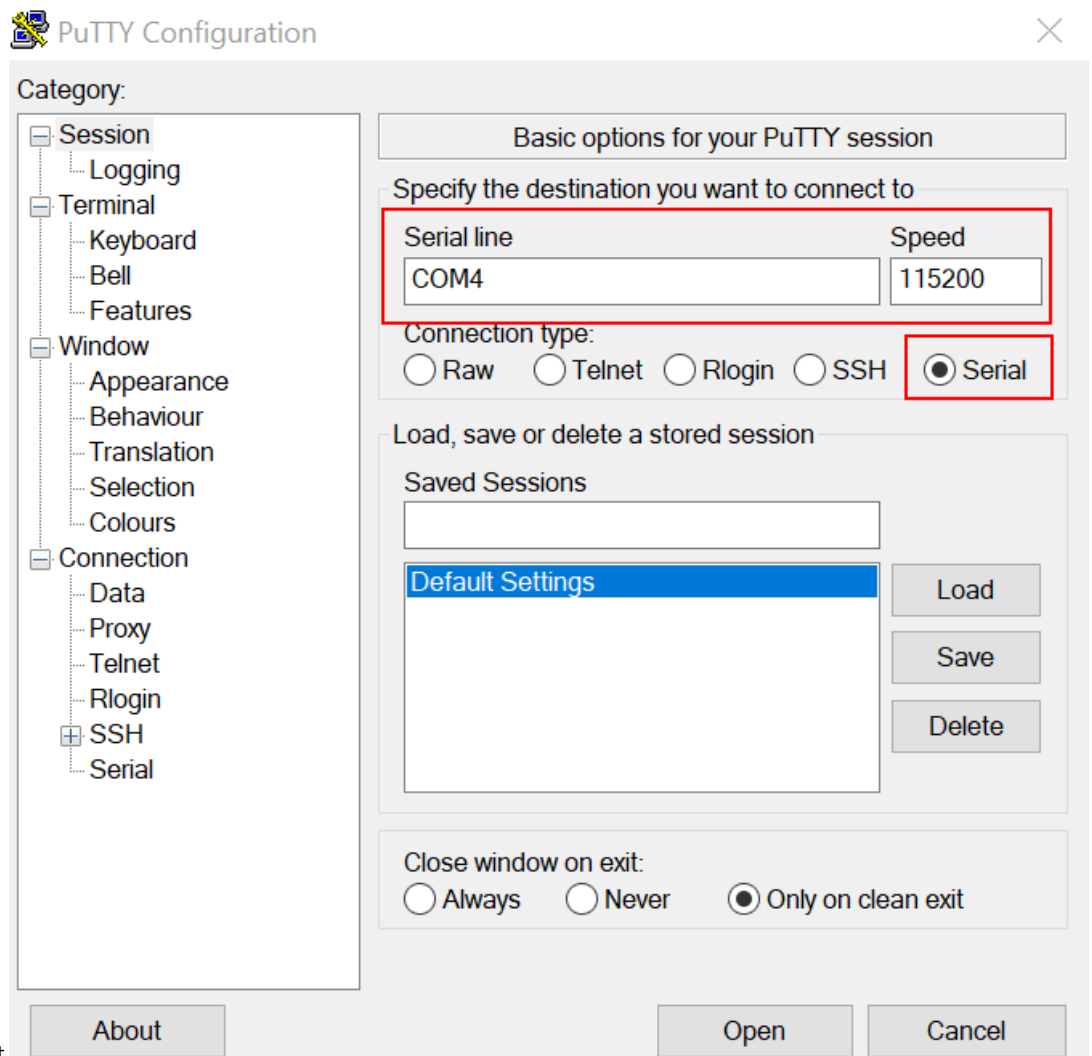
3. To build the demo application, click **Make**, highlighted in red in following figure.



4. The build completes without errors.

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

1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
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 these settings:
 1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
 2. No parity
 3. 8 data bits

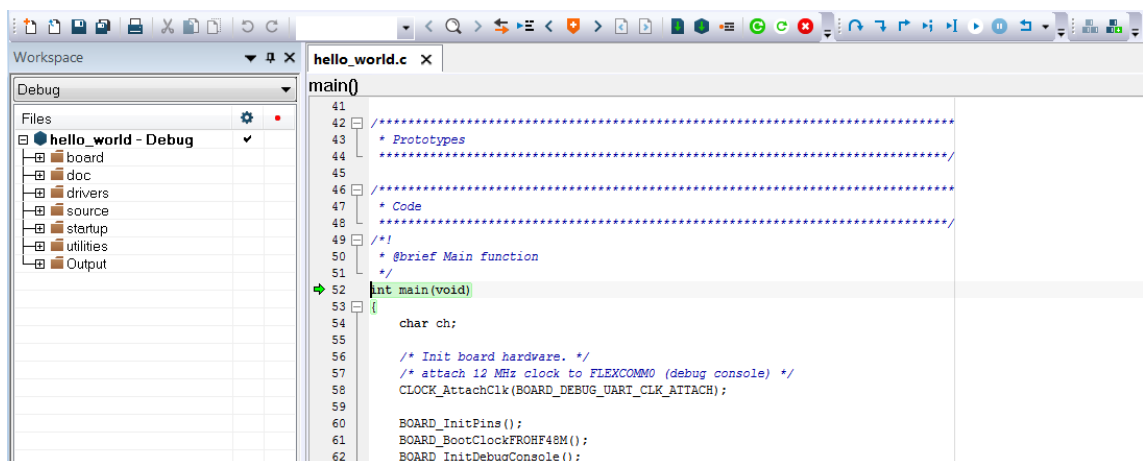


4. 1 stop bit

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



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



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

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.



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

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

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

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/iar/hello_world_cm0plus.  
↪ eww
```

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm0plus) first, because the primary core application project (cm4) must know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

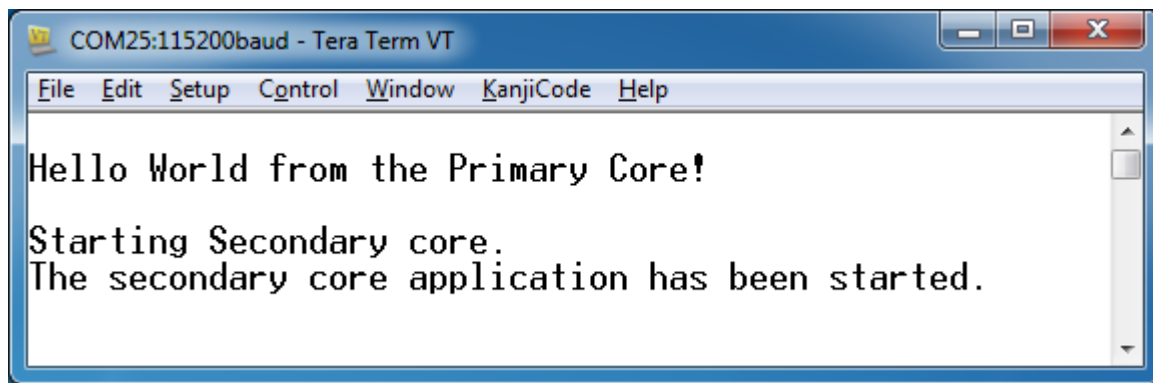
Run a multicore example application The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an example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

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

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



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



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

Build a TrustZone example application This section describes the particular steps that must 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:

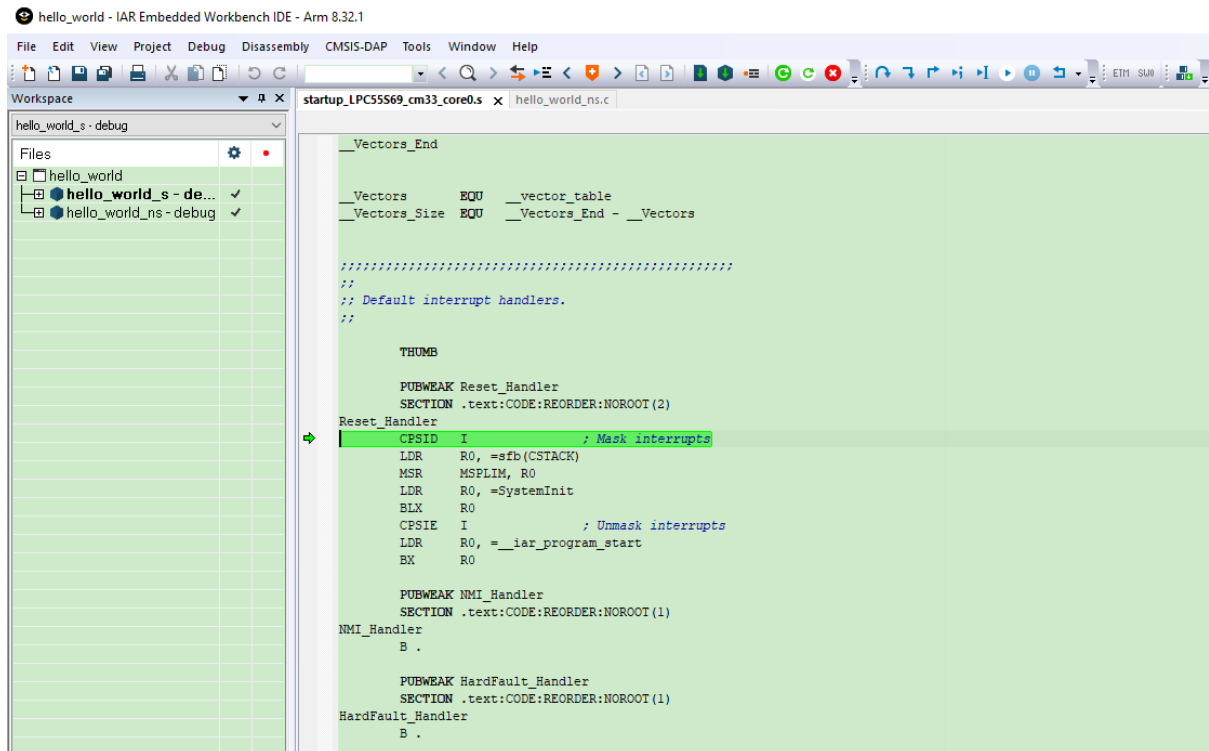
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_  
↪ns.eww
```

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

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

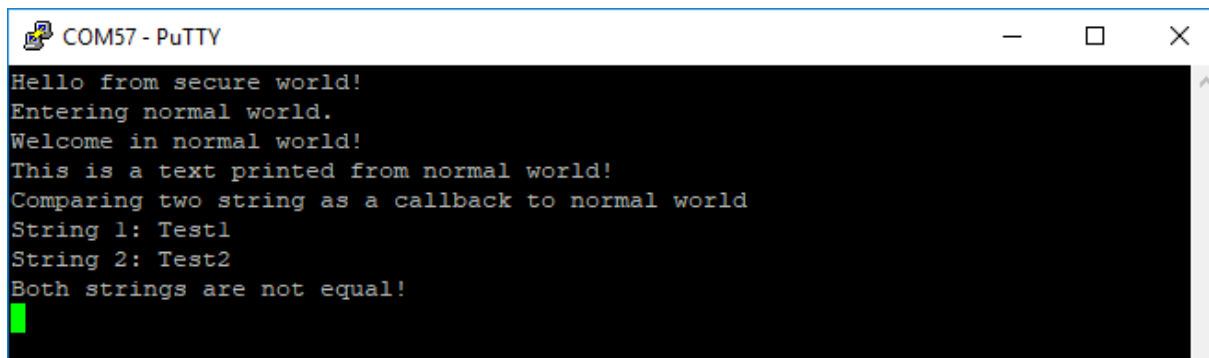
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because the non-secure project must 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.

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 **Run an example application**. These steps are common for both single core, and TrustZone applications in IAR. After clicking **Download and Debug**, both the secure and non-secure images are loaded into the device memory, and the secure application is executed. It stops at the `Reset_Handler` function.

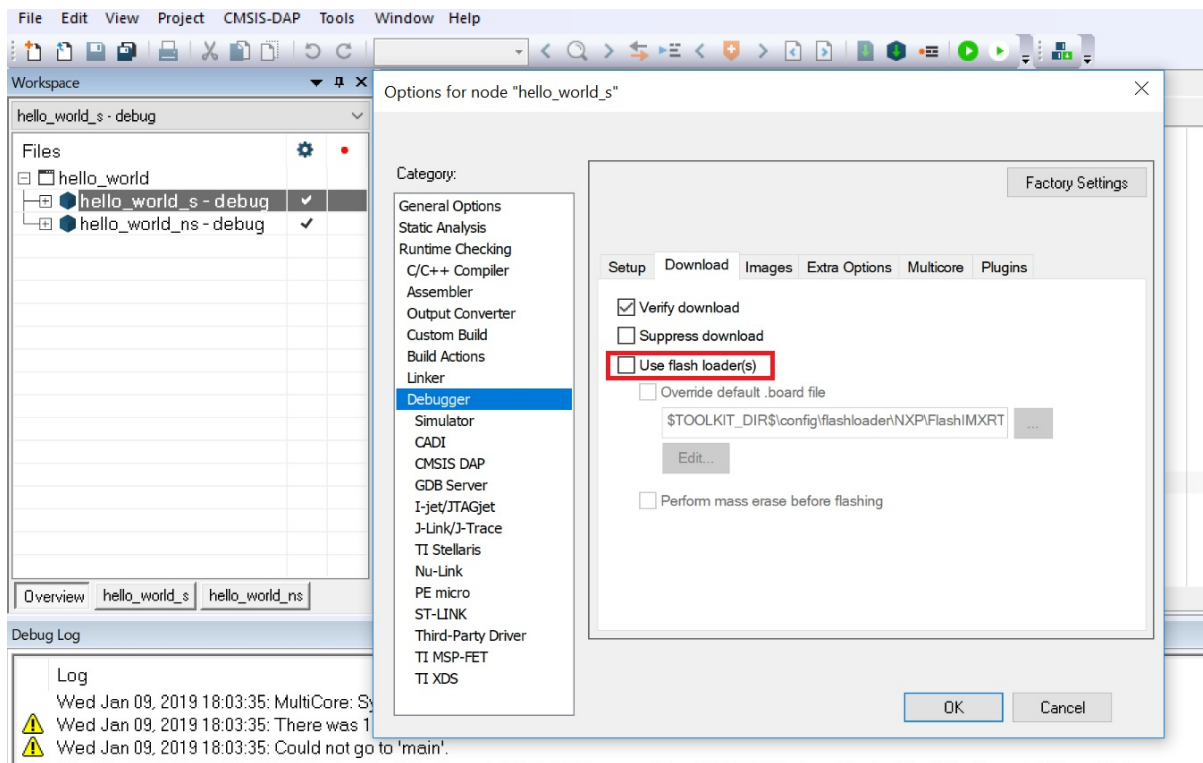


Run the code by clicking **Go** to start the application.

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.



Note: If the application is running in RAM (debug/release build target), in **Options**>**Debugger > Download** tab, disable **Use flash loader(s)**. This can avoid the __ns download issue on i.MXRT500.

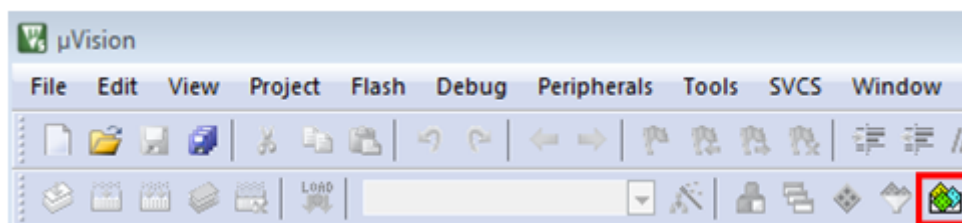


Run a demo using Keil MDK/μVision

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

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

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



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

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:

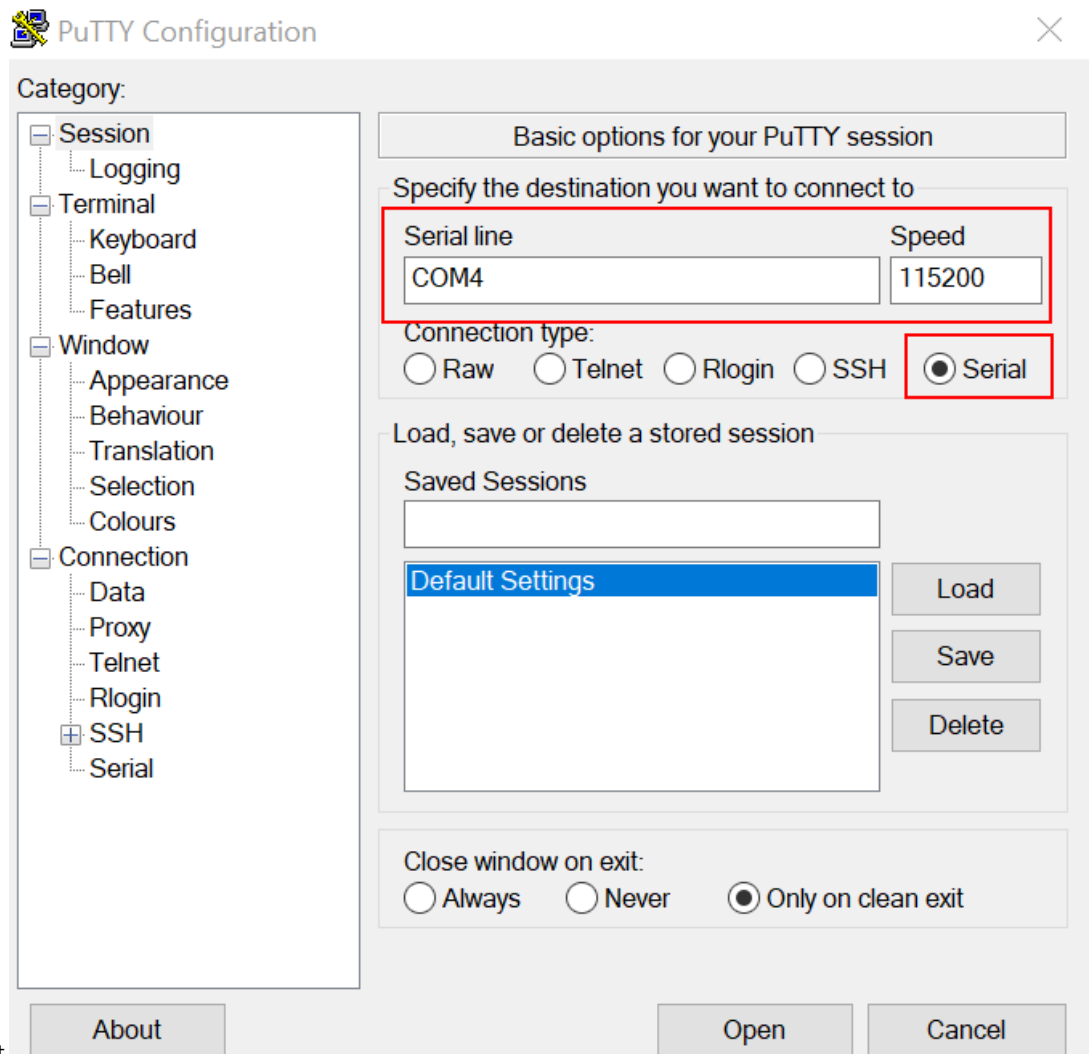
- To build the demo project, select **Rebuild**, highlighted in red.



- The build completes without errors.

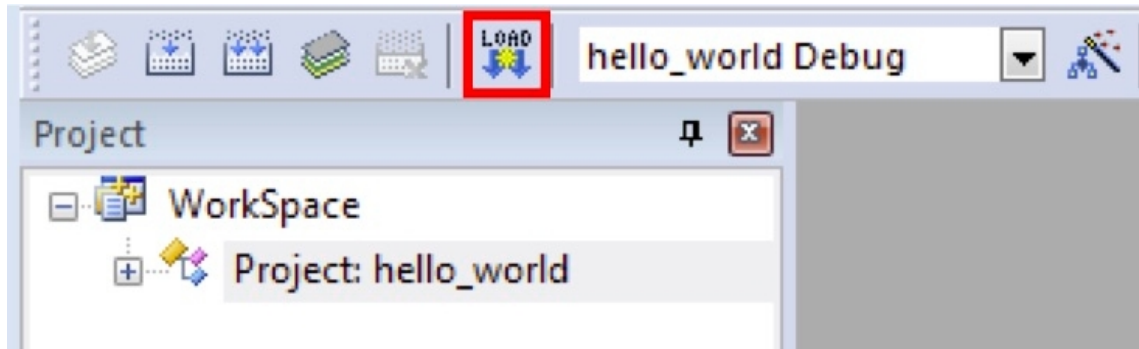
Run an example application To download and run the application, perform these steps:

- Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
- Connect the development platform to your PC via USB cable using USB connector.
- Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
 - 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
 - No parity
 - 8 data bits

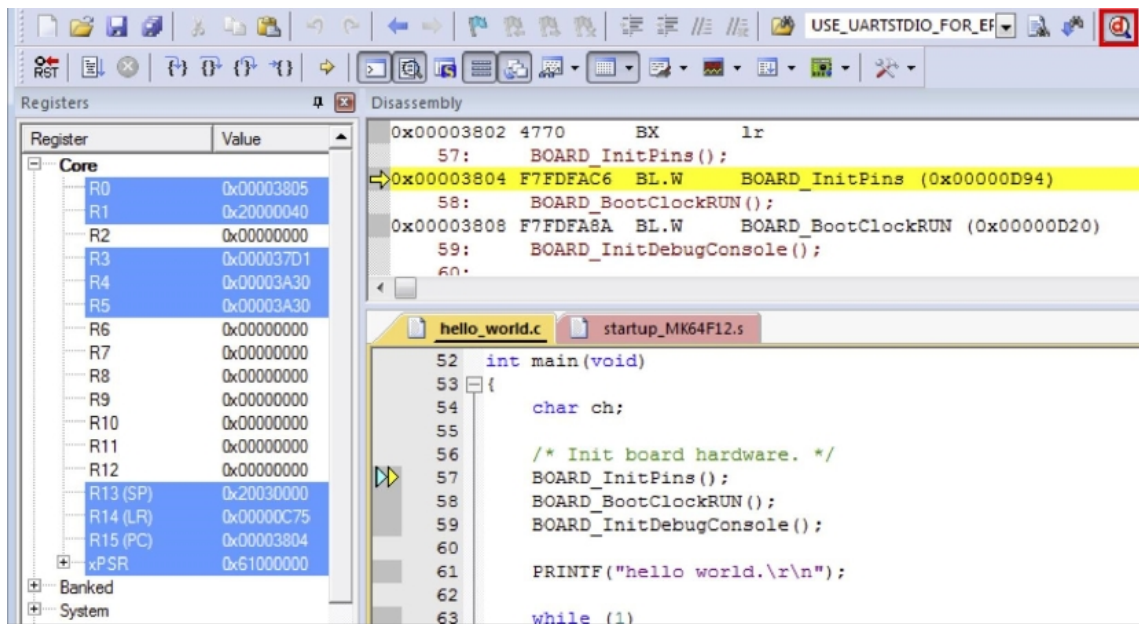


- 1 stop bit

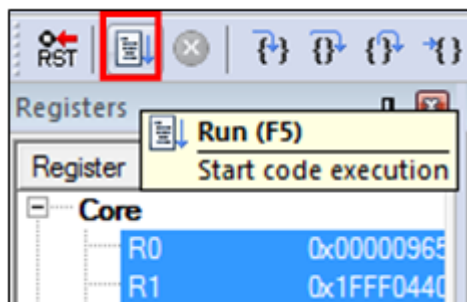
- In μ Vision, after the application is built, click the **Download** button to download the application to the target.



5. After clicking the **Download** button, the application downloads to the target and is running. To debug the application, click the **Start/Stop Debug Session** button, highlighted in red.



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



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

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

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

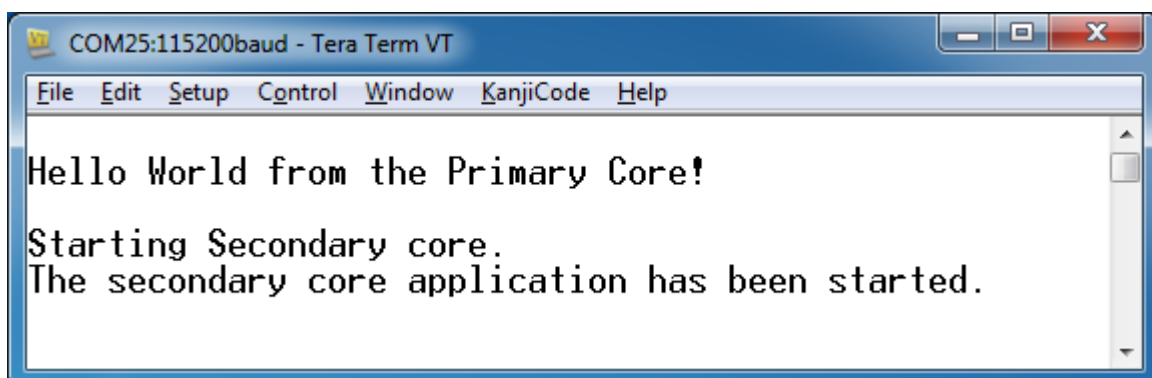
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
→cm0plus.uvmpw
```

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/mdk/hello_world_cm4.uvmpw
```

Build both applications separately by clicking the **Rebuild** button. Build the application for the auxiliary core (cm0plus) first because the primary core application project (cm4) must know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

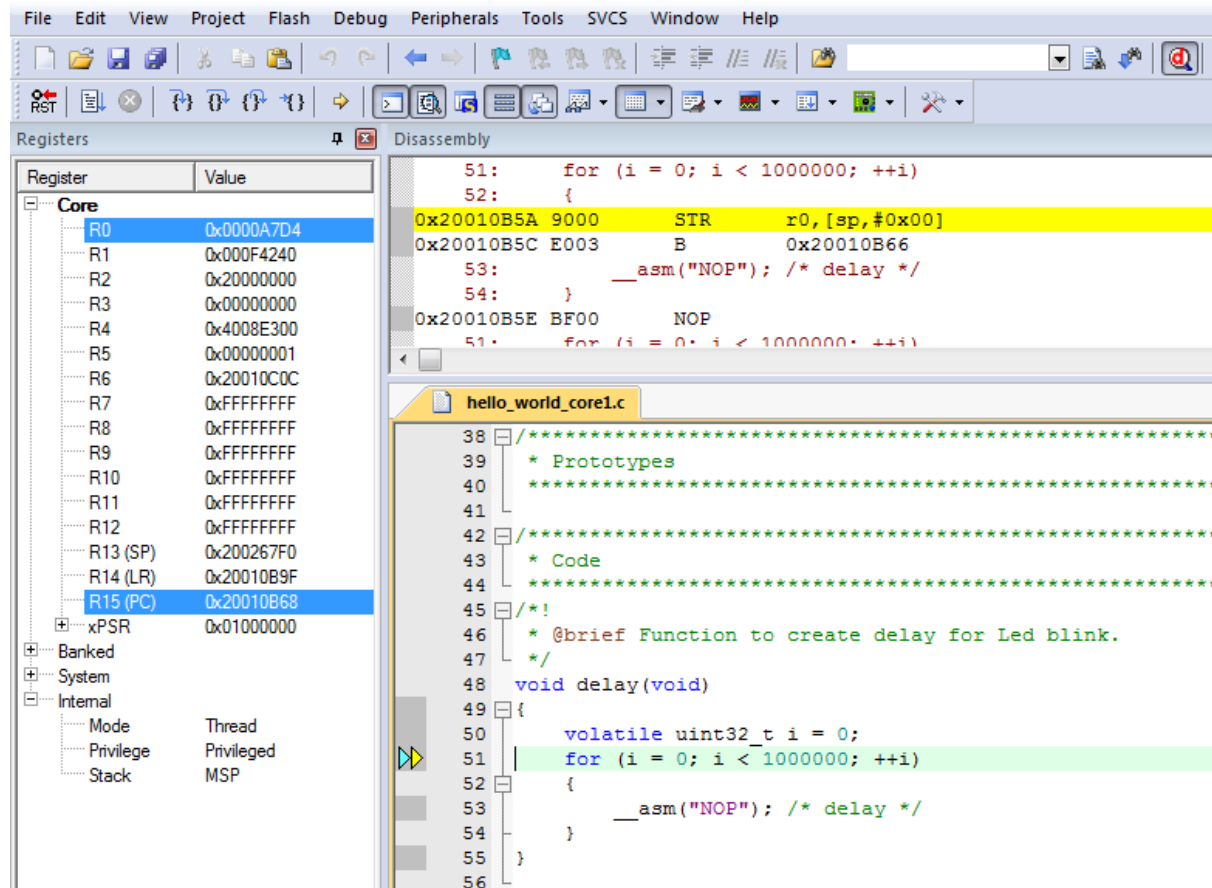
Run a multicore example application The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

Both the primary and the auxiliary image is loaded into the device flash memory. After clicking the “Run” button, the primary core application is executed. During the primary core code execution, the auxiliary core is released from the reset. The hello_world multicore application is now running and a banner is displayed on the terminal. If this does not appear, check your terminal settings and connections.



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

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



Arm describes multicore debugging using the NXP LPC54114 Cortex-M4/M0+ dual-core processor and Keil uVision IDE in Application Note 318 at www.keil.com/appnotes/docs/apnt_318.asp. The associated video can be found [here](#).

Build a TrustZone example application This section describes the particular steps that must 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>/<application_name>_ns/
↪ mdk
```

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<application_name>_s/
↪ mdk
```

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

```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/mdk/hello_world_
↪ ns.uvmpw
```

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



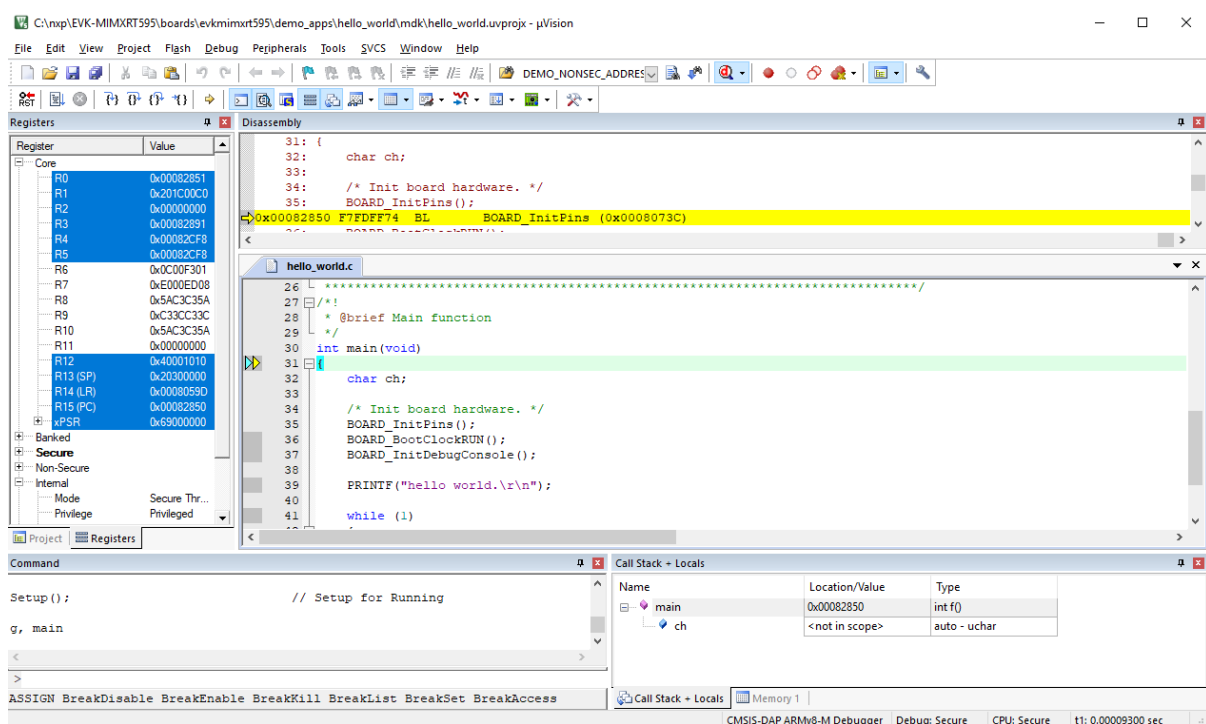
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
→ uvmpw
```

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 must 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.

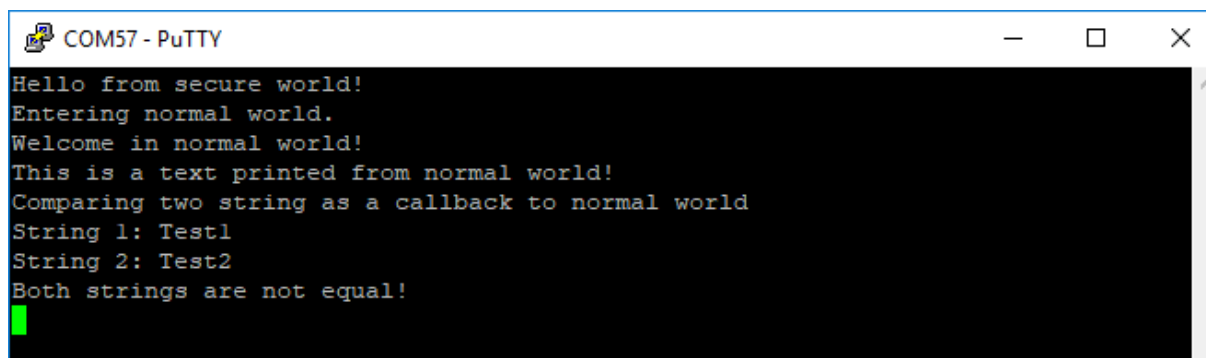
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

To download and run the TrustZone application, switch to the secure application project and perform steps as described in **Run an example application**. These steps are common for single core, dual-core, and TrustZone applications in μ Vision. After clicking **Download and Debug**, both the secure and non-secure images are loaded into the device flash memory, and the secure application is executed. It stops at the `main()` 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 not, check your terminal settings and connections.



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 which is used as an example.

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

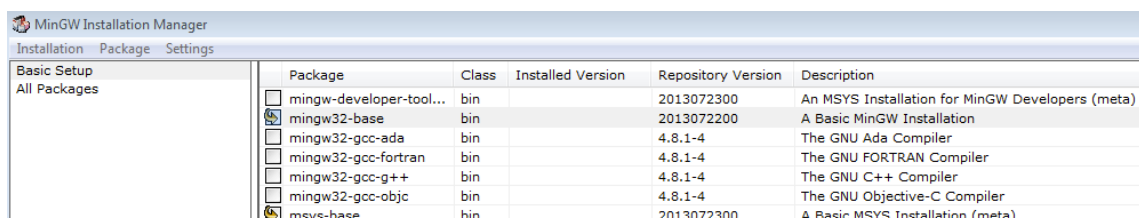
Install GCC Arm Embedded tool chain Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in **MCUXpresso SDK Release Notes**.

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.

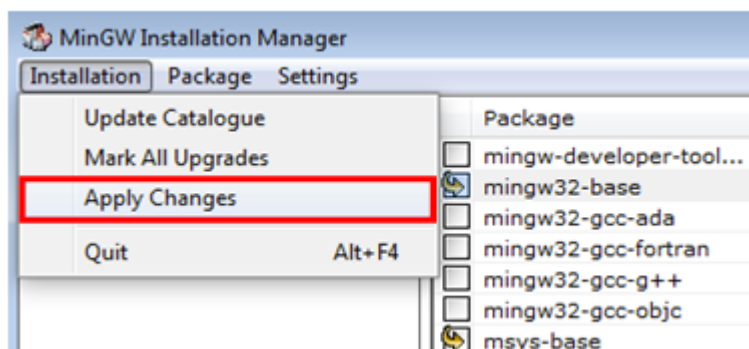
1. Download the latest MinGW mingw-get-setup installer from [MinGW](#).
2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.

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.

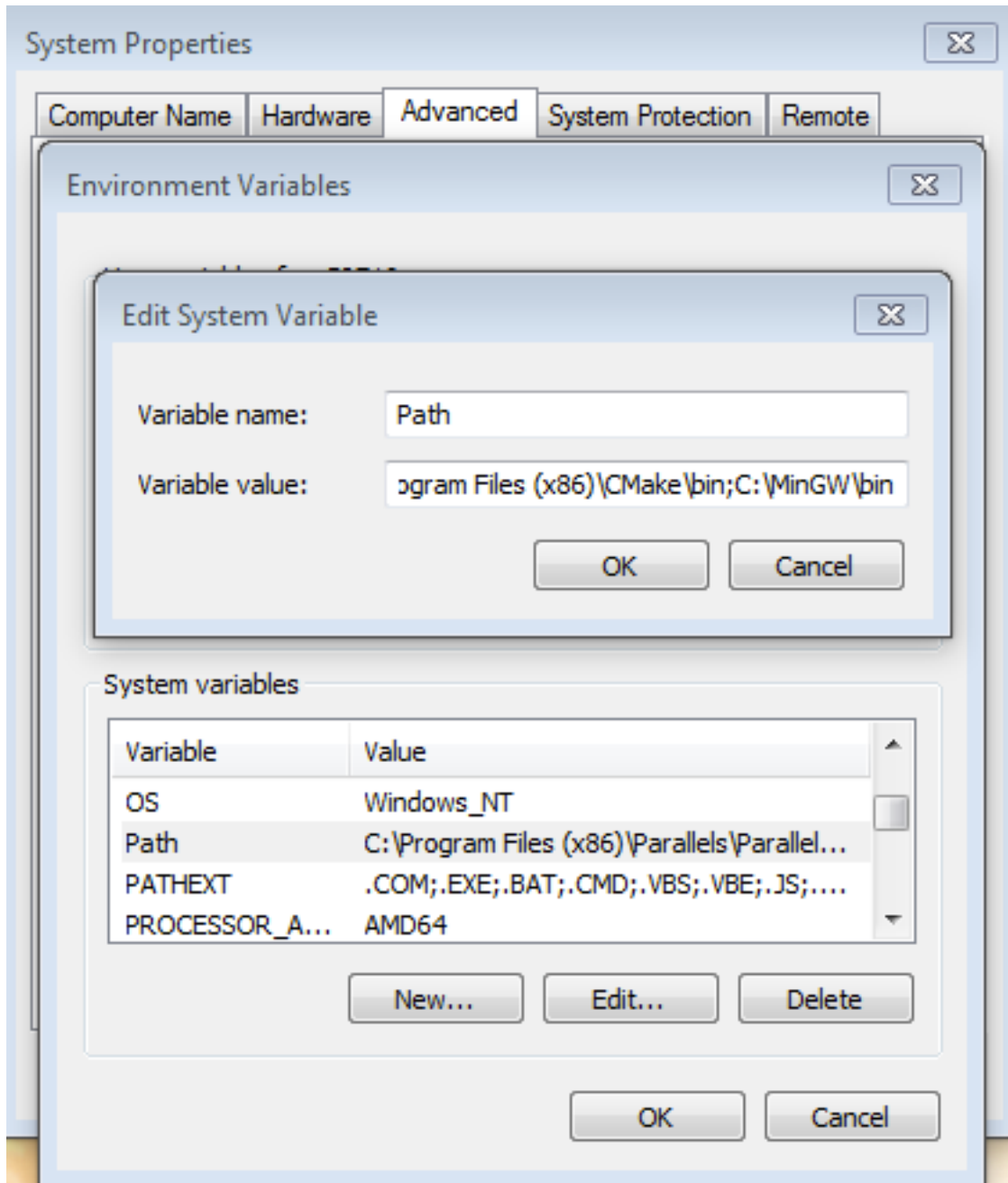


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:


```
<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.



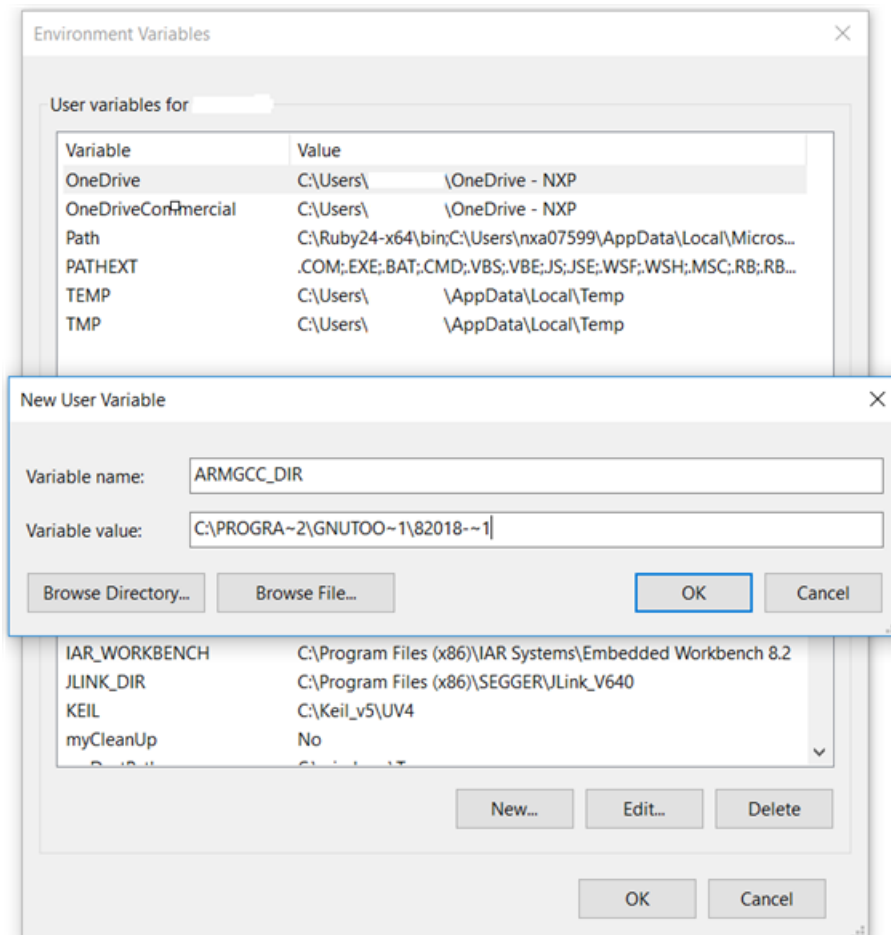
Add a new system environment variable for ARMGCC_DIR Create a new *system* environment variable and name it as ARMGCC_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

```
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 pathname 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

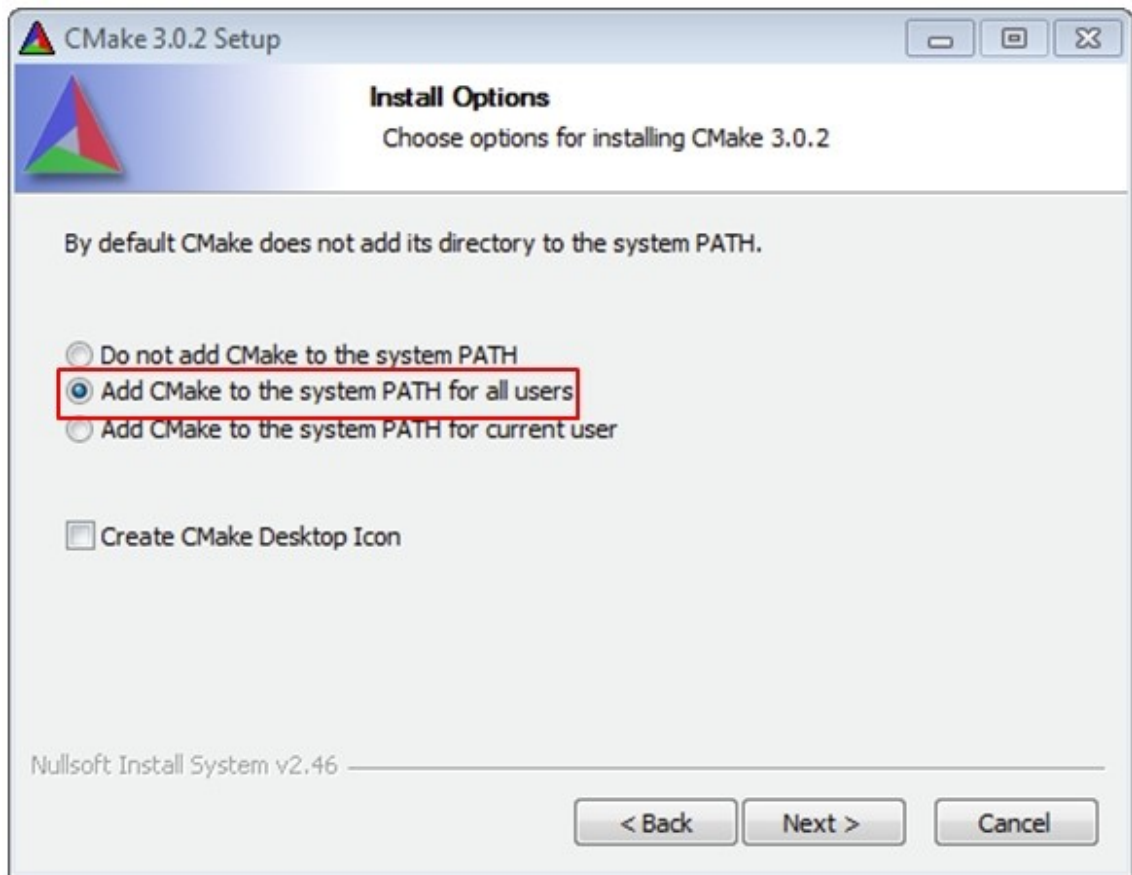
```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```



Install CMake

Windows OS

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.

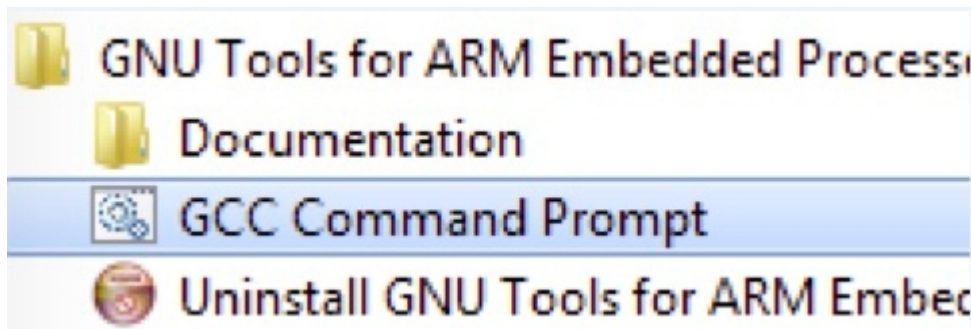
Linux OS It depends on the distributions of Linux Operation System. Here we use Ubuntu as an example.

Open shell and use following commands to install cmake and its version. Ensure the cmake version is above 3.0.x.

```
$ sudo apt-get install cmake
$ cmake --version
```

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:

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 following figure.

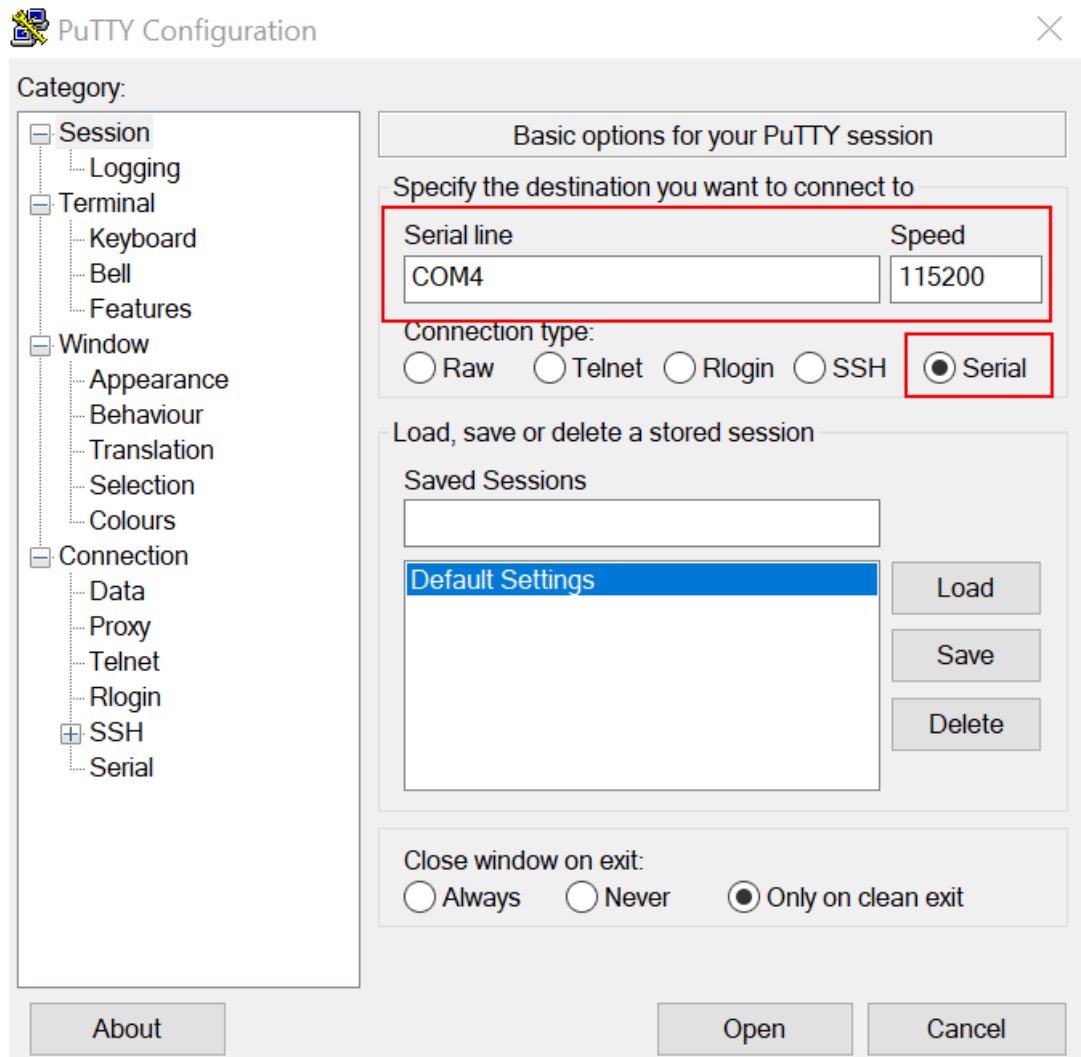
```
[ 84%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_smc.c.obj
[ 92%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.0_FRDM-K64F
/devices/MK64F12/drivers/fsl_clock.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

C:\nxp\SDK_2.0_FRDM-K64F\boards\frdmk64f\demo_apps\hello_world\armgcc>IF "" == "
" <pause >
Press any key to continue . . .
```

Run an example application This section describes steps to run a demo application using J-Link GDB Server application. To install J-Link host driver and update the on-board debugger firmware to Jlink firmware, see [On-board debugger](#).

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 on-board debugger USB connector 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](#)). 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

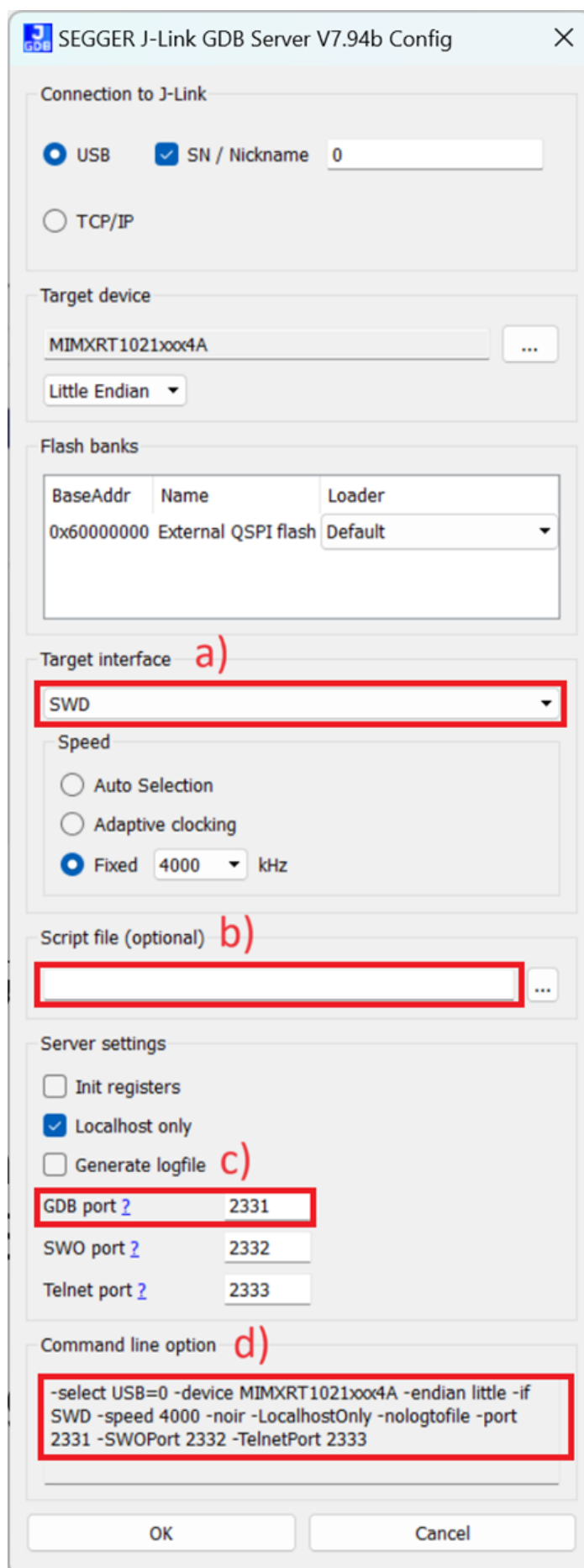


3. To launch the application, open the Windows **Start** menu and select **Programs > SEGGER > J-Link <version> J-Link GDB Server**.

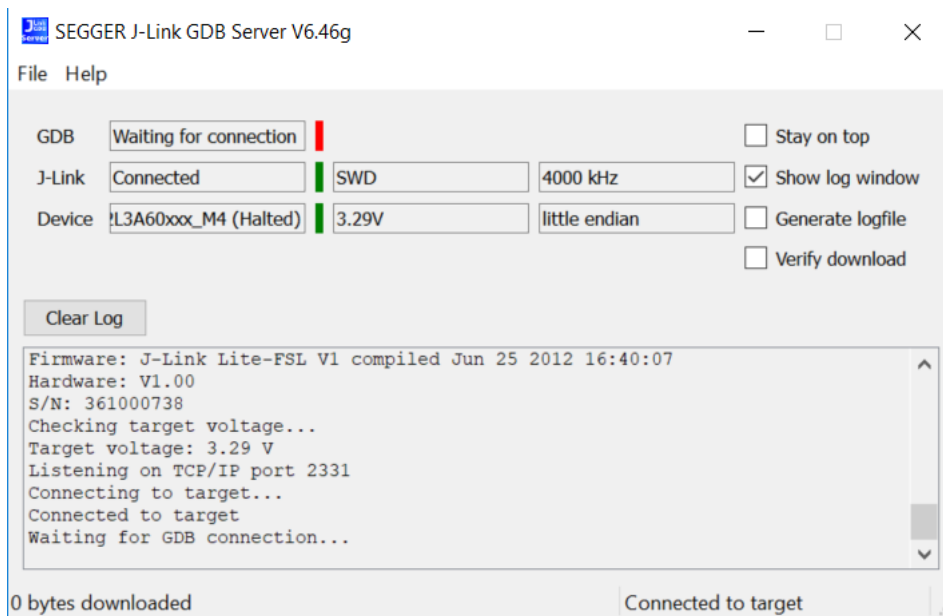
Note: It is assumed that the J-Link software is already installed.

The **SEGGER J-Link GDB Server Config** settings dialog appears.

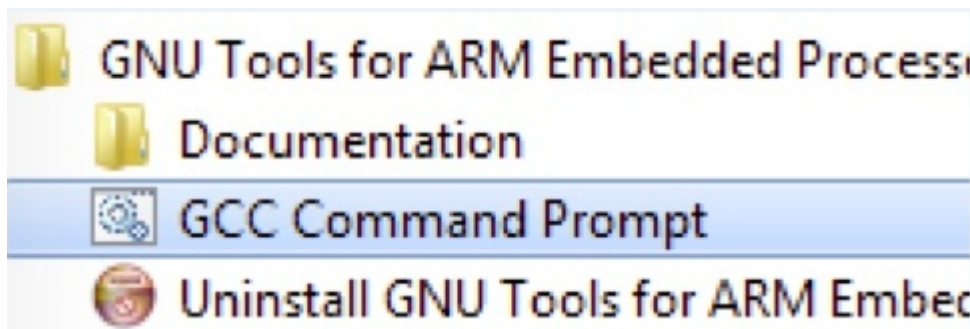
4. Make sure to check the following options.
 1. **Target interface:** The debug connection on board uses internal SWD signaling. In case of a wrong setting J-Link is unable to communicate with device under test.
 2. **Script file:** If required, a J-Link init script file can be used for board initialization. The file with the ".jlinkscript" file extension is located in the <install_dir>/boards/<board_name>/ directory.
 3. Under the **Server settings**, check the GDB port for connection with the gdb target remote command. For more information, see step 9.
 4. There is a command line version of J-Link GDB server "JLinkGDBServerCL.exe". Typical path is C:\Program Files\SEGGER\JLink\. To start the J-Link GDB server with the same settings as are selected in the UI, you can use these command line options.



5. After it is connected, the screen should look like this figure:



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

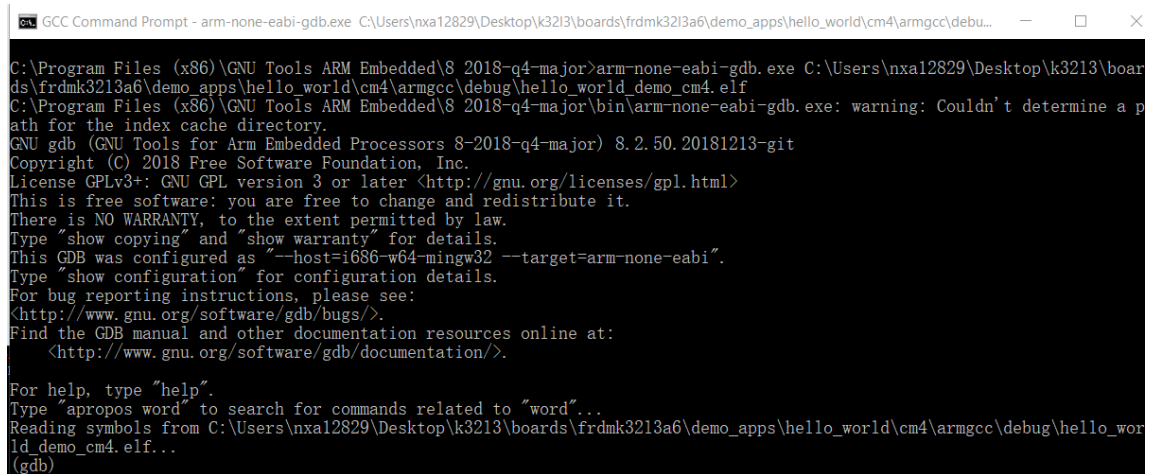


7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

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

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

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



```

C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major>arm-none-eabi-gdb.exe C:\Users\nxa12829\Desktop\k3213\boards\frdmk3213a6\demo_apps\hello_world\cm4\armgcc\debug\hello_world_demo_cm4.elf
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin>arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Tools for Arm Embedded Processors 8-2018-q4-major) 8.2.50.20181213-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"...
Reading symbols from C:\Users\nxa12829\Desktop\k3213\boards\frdmk3213a6\demo_apps\hello_world\cm4\armgcc\debug\hello_world_demo_cm4.elf...
(gdb)

```

9. Run these commands:

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

10. The application is now downloaded and halted. 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.



```

COM4 - PuTTY
hello world.

```

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

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

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

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/armgcc/build_debug.bat
```

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/armgcc/build_debug.bat
```


Build both applications separately following steps for single core examples as described in **Build an example application**.

```

GCC Command Prompt - build_debug.bat

[ 47%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_common.c.obj
[ 52%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_msmc.c.obj
[ 56%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/debug_console/fsl_debug_console.c.obj
[ 60%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/fsl_assert.c.obj
[ 65%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/str/fsl_str.c.obj
[ 69%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/uart/lpuart_adapter.c.obj
[ 73%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_manager.c.obj
[ 78%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_port_uart.c.obj
[ 82%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/lists/generic_list.c.obj
[ 86%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/system_K32L3A60_cm0plus.c.obj
[ 91%] Building ASM object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/gcc/startup_K32L3A60_cm0plus.S.obj
[ 95%] Building C object CMakeFiles/hello_world_cm0plus.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/middleware/multicore/mcmgr/src/mcmgr.c.obj
[100%] Linking C executable debug\hello_world_cm0plus.elf
[100%] Built target hello_world_cm0plus.elf

c:\packages\SDK_2.6.0_FRDM-K32L3A6_RC1\boards\frdmk32l3a6\multicore_examples\hello_world\cm0plus\armgcc>IF "" == "" (pause)
Press any key to continue . . .

GCC Command Prompt - build_debug.bat

[ 50%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_lpuart.c.obj
[ 54%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_common.c.obj
[ 58%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/drivers/fsl_msmc.c.obj
[ 62%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/str/fsl_str.c.obj
[ 66%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/fsl_assert.c.obj
[ 70%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/utilities/debug_console/fsl_debug_console.c.obj
[ 75%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/uart/lpuart_adapter.c.obj
[ 79%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_port_uart.c.obj
[ 83%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/serial_manager/serial_manager.c.obj
[ 87%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/components/lists/generic_list.c.obj
[ 91%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/system_K32L3A60_cm4.c.obj
[ 95%] Building ASM object CMakeFiles/hello_world_cm4.elf.dir/C:/packages/SDK_2.6.0_FRDM-K32L3A6_RC1/devices/K32L3A60/gcc/startup_K32L3A60_cm4.S.obj
[100%] Linking C executable debug\hello_world_cm4.elf
[100%] Built target hello_world_cm4.elf

c:\packages\SDK_2.6.0_FRDM-K32L3A6_RC1\boards\frdmk32l3a6\multicore_examples\hello_world\cm4\armgcc>IF "" == "" (pause)
Press any key to continue . . .

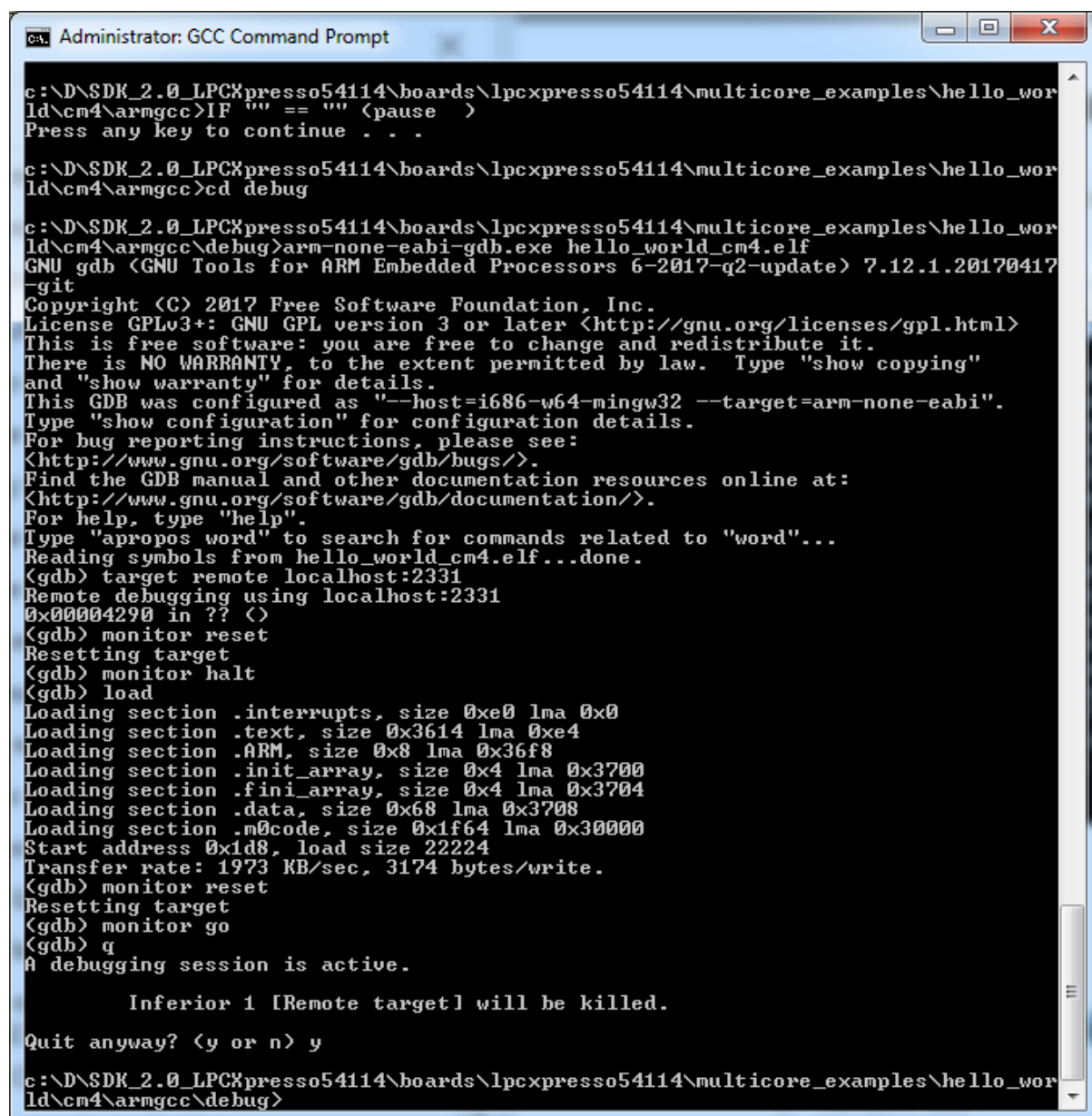
```

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

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

Both the primary and the auxiliary image is loaded into the SPI flash memory. After execution of the monitor go command, the primary core application is executed. During the primary core code execution, the auxiliary core code is reallocated from the flash memory to the RAM, and the auxiliary core is released from the reset. The hello_world multicore application is now running

and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```

Administrator: GCC Command Prompt

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc>IF "" == "" <pause >
Press any key to continue . . .

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc>cd debug

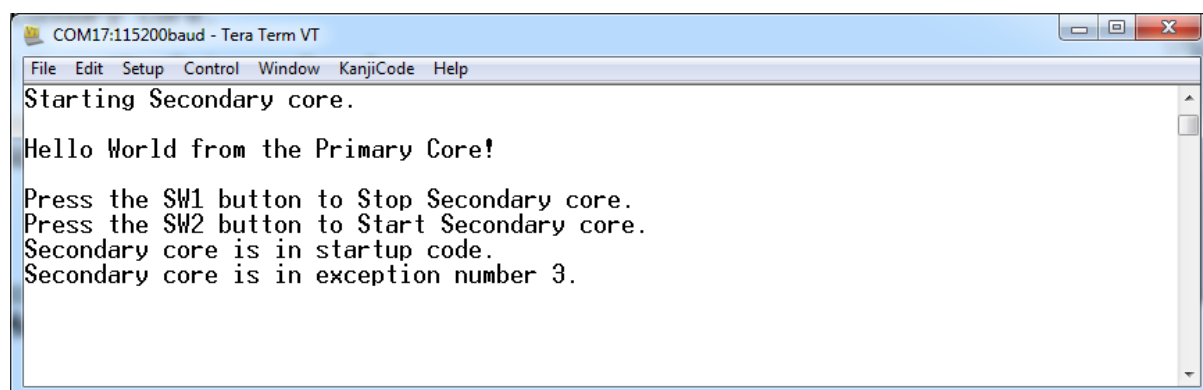
c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc\debug>arm-none-eabi-gdb.exe hello_world_cm4.elf
GNU gdb (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 7.12.1.20170417-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world_cm4.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00004290 in ?? ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0xe0 lma 0x0
Loading section .text, size 0x3614 lma 0xe4
Loading section .ARM, size 0x8 lma 0x36f8
Loading section .init_array, size 0x4 lma 0x3700
Loading section .fini_array, size 0x4 lma 0x3704
Loading section .data, size 0x68 lma 0x3708
Loading section .m0code, size 0x1f64 lma 0x30000
Start address 0x1d8, load size 22224
Transfer rate: 1973 KB/sec, 3174 bytes/write.
(gdb) monitor reset
Resetting target
(gdb) monitor go
(gdb) q
A debugging session is active.

    Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y

c:\D\SDK_2.0_LPCXpresso54114\boards\lpcxpresso54114\multicore_examples\hello_world_cm4\armgcc\debug>

```



```

COM17:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Starting Secondary core.

Hello World from the Primary Core!

Press the SW1 button to Stop Secondary core.
Press the SW2 button to Start Secondary core.
Secondary core is in startup code.
Secondary core is in exception number 3.

```

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>]/  
↪<application_name>_ns/armgcc
```

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/[<core_type>]/  
↪<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/<board_name>/trustzone_examples/hello_world/hello_world_ns/armgcc/build_  
↪debug.bat
```

```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/armgcc/build_  
↪debug.bat
```

Build both applications separately, following steps for single core examples as described in **Build an example application**. It is requested to build the application for the secure project first, because the non-secure project must 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
[ 55%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fsl_
l_assert.c.obj
[ 59%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c
.obj
[ 62%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexspi.c.obj
[ 66%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
cache.c.obj
[ 70%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
l_manager.c.obj
[ 74%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
l_port_uart.c.obj
[ 77%] Building ASM object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startu
p_MIMXRT595S_cm33.S.obj
[ 81%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c
.obj
[ 85%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
usart.c.obj
[ 88%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
iap.c.obj
[100%] Linking C executable debug\hello_world_s.elf
[100%] Built target hello_world_s.elf
C:\npx\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_s\armgcc>IF "" == "" (paus
e)
Press any key to continue . . .
```

```

C:\WINDOWS\system32\cmd.exe
[ 52%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c.obj
[ 56%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fsl_assert.c.obj
[ 60%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexspi.c.obj
[ 64%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_cache.c.obj
[ 68%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_manager.c.obj
[ 72%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_port_uart.c.obj
[ 76%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_usart.c.obj
[ 80%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c.obj
[ 84%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startup_MIMXRT595S_cm33.S.obj
[ 88%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/npx/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_iap.c.obj
[100%] Linking C executable debug\hello_world_ns.elf
[100%] Built target hello_world_ns.elf

C:\npx\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_ns\armgcc>IF "" == "" (pause)
Press any key to continue . . .

```

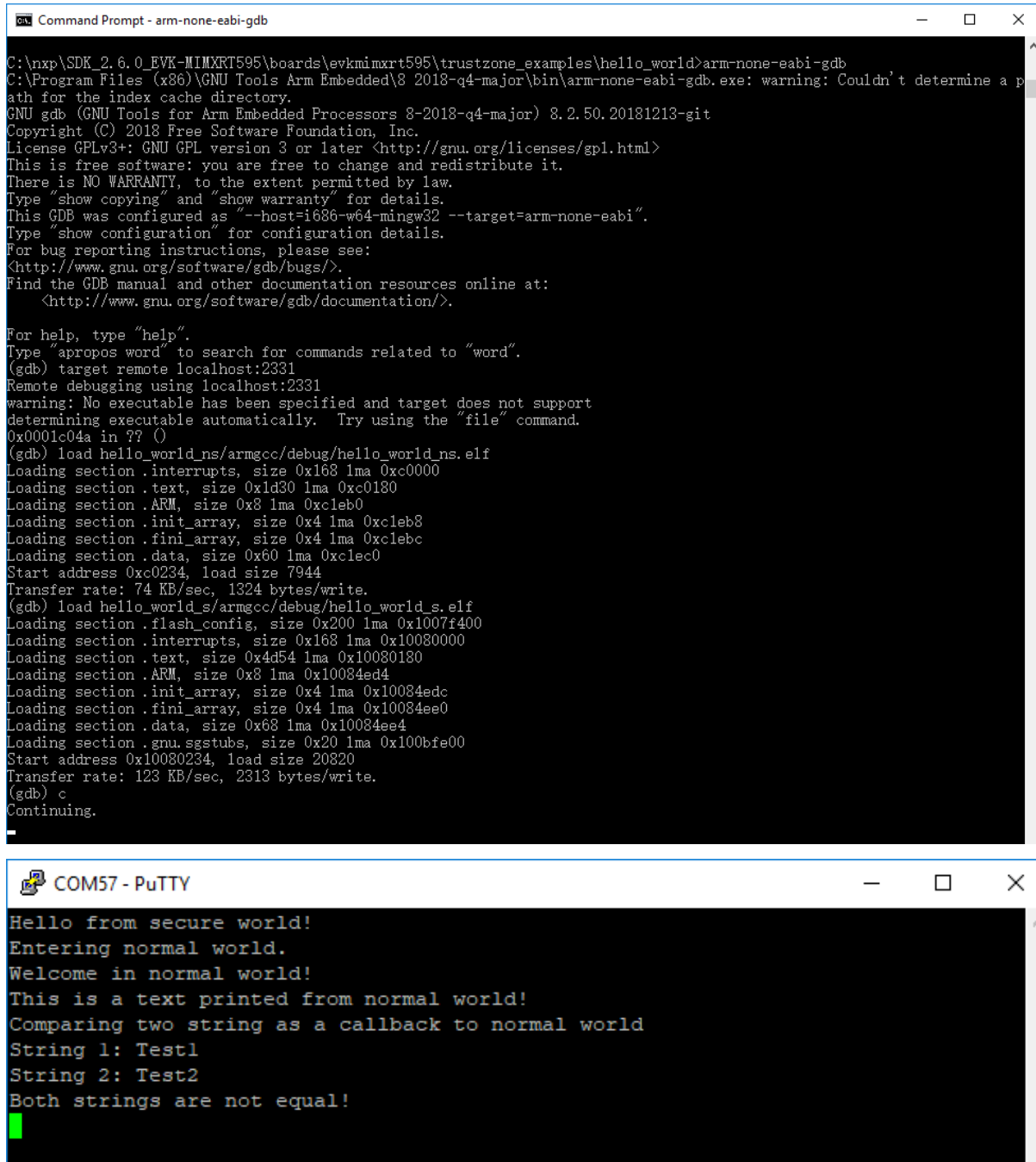
Run a TrustZone example application When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in **Run an example application**.

To download and run the TrustZone application, perform steps 1 to 10, as described in **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. monitor exec SetFlashDLNoRMWThreshold = 0x20000
6. load <install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_ns/armgcc/debug/hello_world_ns.elf
7. load <install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_s/armgcc/debug/hello_world_s.elf
8. monitor reset

The application is now downloaded and halted. Execute the `c` command to start the demo application.



The first screenshot shows a Windows Command Prompt window titled "Command Prompt - arm-none-eabi-gdb". The user has navigated to the directory `C:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world` and executed `arm-none-eabi-gdb`. The prompt shows the GDB version (8.2.50.20181213-git) and target architecture (arm-none-eabi). The user enters `target remote localhost:2331`, and the prompt shows the target is remote. The user then enters `load hello_world_ns/armgcc/debug/hello_world_ns.elf`, and the prompt shows the file is loaded. The user enters `c` to continue, and the prompt shows the program is running.

The second screenshot shows a PuTTY window titled "COM57 - PuTTY". The output of the program is displayed in the terminal window. The output is as follows:

```

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!

```

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 24.12.00 or later.

Following table 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 set up various on-chip peripherals prior to 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 μ Vision, or Arm GCC.
- **Online version** available on mcuxpresso.nxp.com. Recommended doing a quick evaluation of the processor or use the tool without installation.

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

How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform. All NXP boards ship with a factory programmed, onboard debug interface, whether it is based on MCU-Link or the legacy OpenSDA, LPC-Link2, 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 for core0 debug console and the other is for core1.

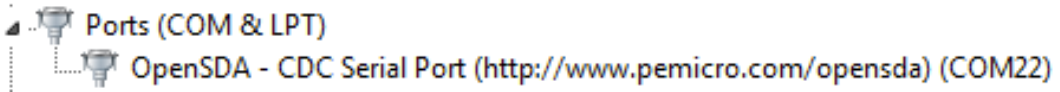
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names are different for all the NXP boards.

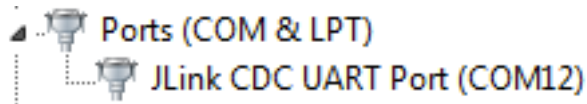
1. **CMSIS-DAP/mbd/DAPLink** interface:



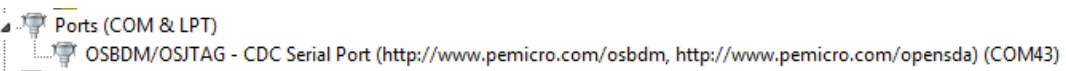
2. P&E Micro:



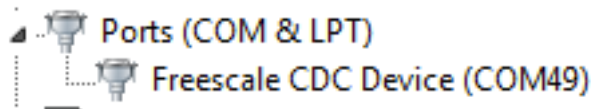
3. J-Link:



4. P&E Micro OSJTAG:



5. MRB-KW01:



On-board Debugger

This section describes the on-board debuggers used on NXP development boards.

On-board debugger MCU-Link MCU-Link is a powerful and cost effective debug probe that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol. MCU-Link also includes a USB to UART bridge feature (VCOM) that can be used to provide a serial connection between the target MCU and a host computer. MCU-Link features a high-speed USB interface for high performance debug. MCU-Link is compatible with Windows, MacOS and Linux. A free utility from NXP provides an easy way to install firmware updates.

On-board MCU-Link debugger supports CMSIS-DAP and J-Link firmware. See the table in [Default debug interfaces](#) to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, 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.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.

Updating MCU-Link firmware This firmware in this debug interface may be updated using the host computer utility called MCU-Link. 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 reprogram 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), MCU-Link 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 that most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the MCU-Link utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link or NXP boards. The utility can be downloaded from [MCU-Link](#).

These steps show how to update the debugger firmware on your board for Windows operating system.

1. Install the MCU-Link utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labeled 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 MCU-Link installation directory (<MCU-Link install dir>).
 1. To program CMSIS-DAP debug firmware: <MCU-Link install dir>/scripts/program_CMSIS
 2. To program J-Link debug firmware: <MCU-Link install dir>/scripts/program_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

On-board debugger LPC-Link LPC-Link 2 is an extensible debug probe that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol. MCU-Link also includes a USB to UART bridge feature (VCOM) that can be used to provide a serial connection between the target MCU and a host computer. LPC-Link 2 is compatible with Windows, MacOS and Linux. A free utility from NXP provides an easy way to install firmware updates.

On-board LPC-Link 2 debugger supports CMSIS-DAP and J-Link firmware. See the table in [Default debug interfaces](#) to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, 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.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.

Updating LPC-Link 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 reprogram 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 that 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 [LPCScript](#).

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 ([LPCScript](#), 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 labeled 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. Repower the board by removing the USB cable and plugging it in again.

On-board debugger OpenSDA OpenSDA/OpenSDAv2 is a serial and debug adapter that is built into several NXP evaluation boards. It provides a bridge between your computer (or other USB host) and the embedded target processor, which can be used for debugging, flash programming, and serial communication, all over a simple USB cable.

The difference is the firmware implementation: OpenSDA: Programmed with the proprietary P&E Micro developed bootloader. P&E Micro is the default debug interface app. OpenSDAv2: Programmed with the open-sourced CMSIS-DAP/mbed bootloader. CMSIS-DAP is the default debug interface app.

See the table in [Default debug interfaces](#) to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, 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 a P&E Micro interface, see [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

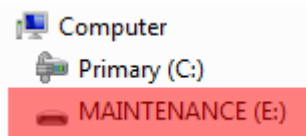
Updating OpenSDA firmware Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. This typically means to switch from the default application (either CMSIS-DAP or P&E Micro) to a SEGGER J-Link. This section contains the steps to switch the OpenSDA firmware to a J-Link interface. However, the steps can be applied to restoring the original image also. For reference, OpenSDA firmware files can be found at the links below:

- J-Link: Download appropriate image from www.segger.com/opensda.html. Choose the appropriate J-Link binary based on the table in [Default debug interfaces](#). Any OpenSDA v1.0 interface should use the standard OpenSDA download (in other words, the one with no version). For OpenSDA 2.0 or 2.1, select the corresponding binary.
- CMSIS-DAP: CMSIS-DAP OpenSDA firmware is available at www.nxp.com/opensda.

- P&E Micro: Downloading P&E Micro OpenSDA firmware images requires registration with P&E Micro (www.pemicro.com).

Perform the following steps to update the OpenSDA firmware on your board for Windows and Linux OS users:

1. Unplug the board's USB cable.
2. Press the **Reset** button on the board. While still holding the button, plug the USB cable back into the board.
3. When the board re-enumerates, it shows up as a disk drive called **MAINTENANCE**.



4. Drag and drop the new firmware image onto the MAINTENANCE drive.

Note: If for any reason the firmware update fails, the board can always reenter maintenance mode by holding down **Reset** button and power cycling.

These steps show how to update the OpenSDA firmware on your board for Mac OS users.

1. Unplug the board's USB cable.
2. Press the **Reset** button of the board. While still holding the button, plug the USB cable back into the board.
3. For boards with OpenSDA v2.0 or v2.1, it shows up as a disk drive called **BOOTLOADER** in **Finder**. Boards with OpenSDA v1.0 may or may not show up depending on the bootloader version. If you see the drive in **Finder**, proceed to the next step. If you do not see the drive in **Finder**, use a PC with Windows OS 7 or an earlier version to either update the OpenSDA firmware, or update the OpenSDA bootloader to version 1.11 or later. The bootloader update instructions and image can be obtained from P&E Microcomputer website.
4. For OpenSDA v2.1 and OpenSDA v1.0 (with bootloader 1.11 or later) users, drag the new firmware image onto the BOOTLOADER drive in **Finder**.
5. For OpenSDA v2.0 users, type these commands in a Terminal window:

```
> sudo mount -u -w -o sync /Volumes/BOOTLOADER  
> cp -X <path to update file> /Volumes/BOOTLOADER
```

Note: If for any reason the firmware update fails, the board can always reenter bootloader mode by holding down the **Reset** button and power cycling.

On-board debugger Multilink An on-board Multilink debug circuit provides a JTAG interface and a power supply input through a single micro-USB connector. It is a hardware interface that allows PC software to debug and program a target processor through its debug port.

The host driver must be installed before debugging.

- See [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

On-board debugger OSJTAG An on-board OSJTAG debug circuit provides a JTAG interface and a power supply input through a single micro-USB connector. It is a hardware interface that allows PC software to debug and program a target processor through its debug port.

The host driver must be installed before debugging.

- See [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

Default debug interfaces

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

Hardware platform	Default debugger firmware	On-board debugger probe
EVK-MCIMX7ULP	N/A	N/A
EVK-MIMX8MM	N/A	N/A
EVK-MIMX8MN	N/A	N/A
EVK-MIMX8MNDDR3L	N/A	N/A
EVK-MIMX8MP	N/A	N/A
EVK-MIMX8MQ	N/A	N/A
EVK-MIMX8ULP	N/A	N/A
EVK-MIMXRT1010	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1015	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1020	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1064	CMSIS-DAP	LPC-Link2
EVK-MIMXRT595	CMSIS-DAP	LPC-Link2
EVK-MIMXRT685	CMSIS-DAP	LPC-Link2
EVK9-MIMX8ULP	N/A	N/A
EVKB-IMXRT1050	CMSIS-DAP	LPC-Link2
FRDM-K22F	CMSIS-DAP	OpenSDA v2
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2
FRDM-K32L3A6	CMSIS-DAP	OpenSDA v2
FRDM-KE02Z40M	P&E Micro	OpenSDA v1
FRDM-KE15Z	CMSIS-DAP	OpenSDA v2
FRDM-KE16Z	CMSIS-DAP	OpenSDA v2
FRDM-KE17Z	CMSIS-DAP	OpenSDA v2
FRDM-KE17Z512	CMSIS-DAP	MCU-Link
FRDM-MCXA153	CMSIS-DAP	MCU-Link
FRDM-MCXA156	CMSIS-DAP	MCU-Link
FRDM-MCXA266	CMSIS-DAP	MCU-Link
FRDM-MCXA344	CMSIS-DAP	MCU-Link
FRDM-MCXA346	CMSIS-DAP	MCU-Link
FRDM-MCXA366	CMSIS-DAP	MCU-Link
FRDM-MCXC041	CMSIS-DAP	MCU-Link
FRDM-MCXC242	CMSIS-DAP	MCU-Link
FRDM-MCXC444	CMSIS-DAP	MCU-Link
FRDM-MCXE247	CMSIS-DAP	MCU-Link
FRDM-MCXE31B	CMSIS-DAP	MCU-Link
FRDM-MCXN236	CMSIS-DAP	MCU-Link
FRDM-MCXN947	CMSIS-DAP	MCU-Link
FRDM-MCXW23	CMSIS-DAP	MCU-Link
FRDM-MCXW71	CMSIS-DAP	MCU-Link
FRDM-MCXW72	CMSIS-DAP	MCU-Link
FRDM-RW612	CMSIS-DAP	MCU-Link
IMX943-EVK	N/A	N/A
IMX95LP4XEVK-15	N/A	N/A
IMX95LPD5EVK-19	N/A	N/A
IMX95VERDINEVK	N/A	N/A
KW45B41Z-EVK	CMSIS-DAP	MCU-Link
KW45B41Z-LOC	CMSIS-DAP	MCU-Link
KW47-EVK	CMSIS-DAP	MCU-Link

continues on next page

Table 1 – continued from previous page

Hardware platform	Default debugger firmware	On-board debugger probe
KW47-LOC	CMSIS-DAP	MCU-Link
LPC845BREAKOUT	CMSIS-DAP	LPC-Link2
LPCXpresso51U68	CMSIS-DAP	LPC-Link2
LPCXpresso54628	CMSIS-DAP	LPC-Link2
LPCXpresso54S018	CMSIS-DAP	LPC-Link2
LPCXpresso54S018M	CMSIS-DAP	LPC-Link2
LPCXpresso55S06	CMSIS-DAP	LPC-Link2
LPCXpresso55S16	CMSIS-DAP	LPC-Link2
LPCXpresso55S28	CMSIS-DAP	LPC-Link2
LPCXpresso55S36	CMSIS-DAP	MCU-Link
LPCXpresso55S69	CMSIS-DAP	LPC-Link2
LPCXpresso802	CMSIS-DAP	LPC-Link2
LPCXpresso804	CMSIS-DAP	LPC-Link2
LPCXpresso824MAX	CMSIS-DAP	LPC-Link2
LPCXpresso845MAX	CMSIS-DAP	LPC-Link2
LPCXpresso860MAX	CMSIS-DAP	LPC-Link2
MC56F80000-EVK	P&E Micro	Multilink
MC56F81000-EVK	P&E Micro	Multilink
MC56F83000-EVK	P&E Micro	OSJTAG
MCIMX93-EVK	N/A	N/A
MCIMX93-QSB	N/A	N/A
MCIMX93AUTO-EVK	N/A	N/A
MCX-N5XX-EVK	CMSIS-DAP	MCU-Link
MCX-N9XX-EVK	CMSIS-DAP	MCU-Link
MCX-W71-EVK	CMSIS-DAP	MCU-Link
MCX-W72-EVK	CMSIS-DAP	MCU-Link
MIMXRT1024-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1040-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1060-EVKB	CMSIS-DAP	LPC-Link2
MIMXRT1060-EVKC	CMSIS-DAP	MCU-Link
MIMXRT1160-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1170-EVKB	CMSIS-DAP	MCU-Link
MIMXRT1180-EVK	CMSIS-DAP	MCU-Link
MIMXRT685-AUD-EVK	CMSIS-DAP	LPC-Link2
MIMXRT700-EVK	CMSIS-DAP	MCU-Link
RD-RW612-BGA	CMSIS-DAP	MCU-Link
TWR-KM34Z50MV3	P&E Micro	OpenSDA v1
TWR-KM34Z75M	P&E Micro	OpenSDA v1
TWR-KM35Z75M	CMSIS-DAP	OpenSDA v2
TWR-MC56F8200	P&E Micro	OSJTAG
TWR-MC56F8400	P&E Micro	OSJTAG

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
}
```

1.3 Getting Started with MCUXpresso SDK GitHub

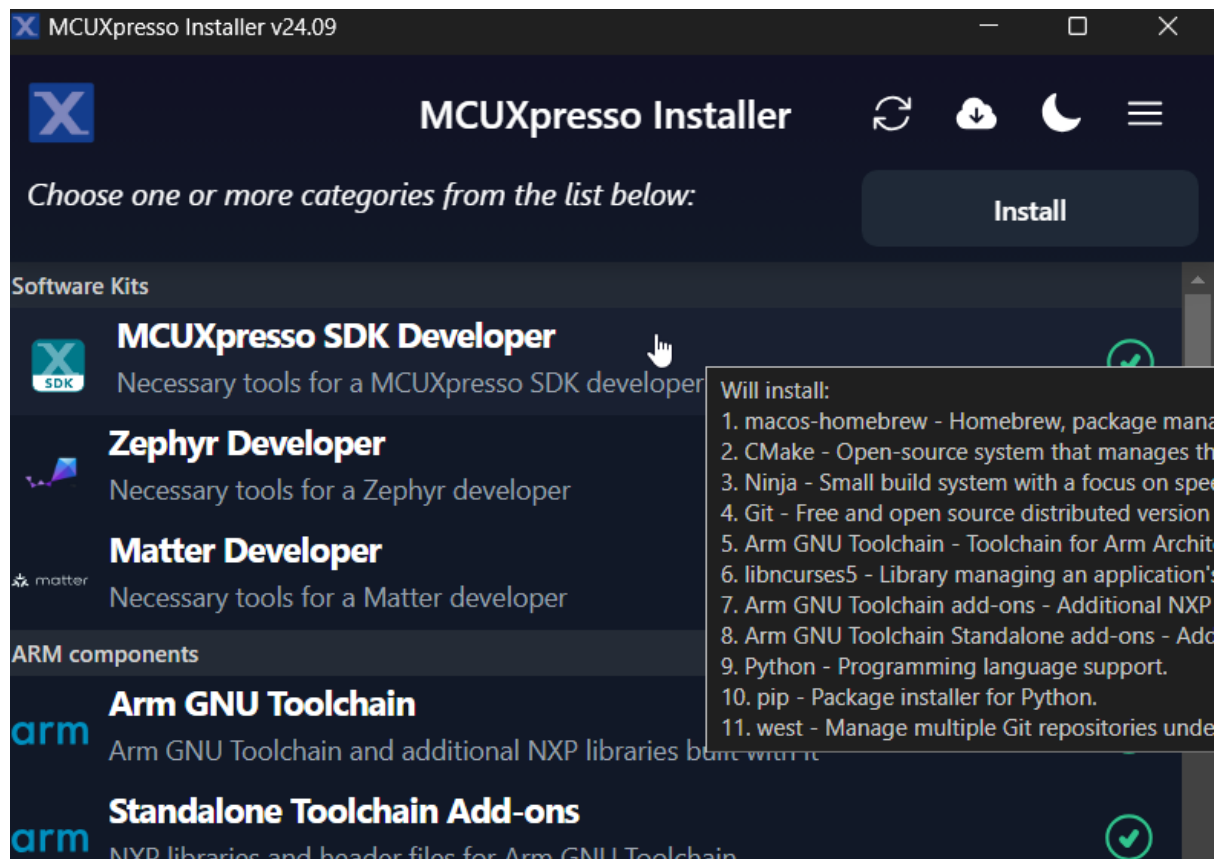
1.3.1 Getting Started with MCUXpresso SDK Repository

Installation

NOTE

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

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



Alternative: Manual Installation

Basic tools

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

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

Then configure your username and email using the commands:

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

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

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

West Please use the west version equal or greater than 1.2.0

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

Build And Configuration System

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

For Windows, you can directly use the .msi installer like [cmake-3.31.4-windows-x86_64.msi](#) to install.

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

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

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

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

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

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

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

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

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

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

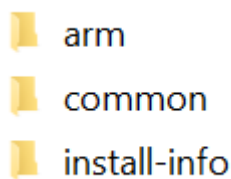
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

```
cd mcuxsdk
```

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

```
python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

Explore Contents

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

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

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

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

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

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

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

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

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

Run a demo using MCUXpresso for VS Code

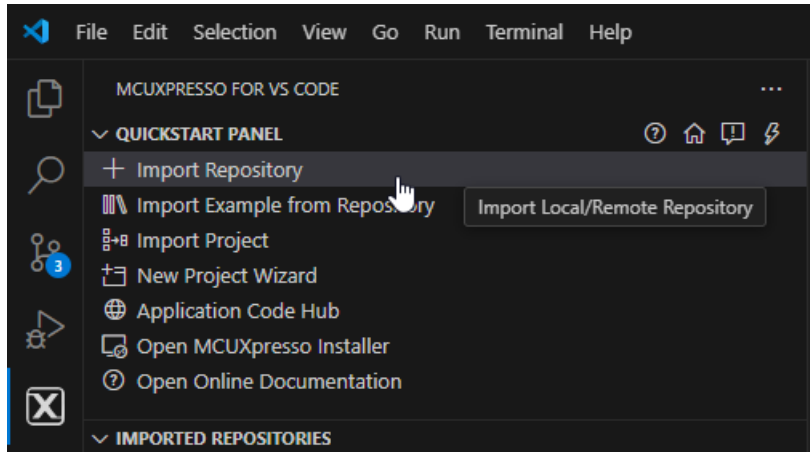
This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the `hello_world` demo application as an example. However, these

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

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

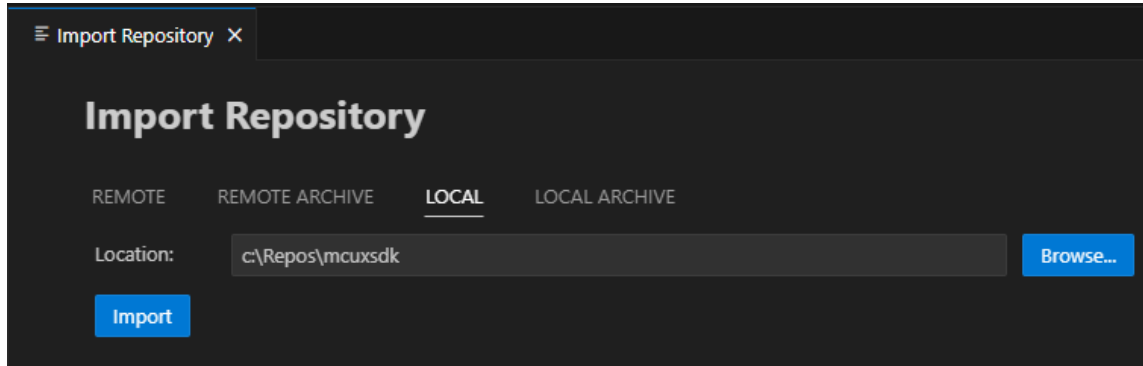
To build an example application:

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

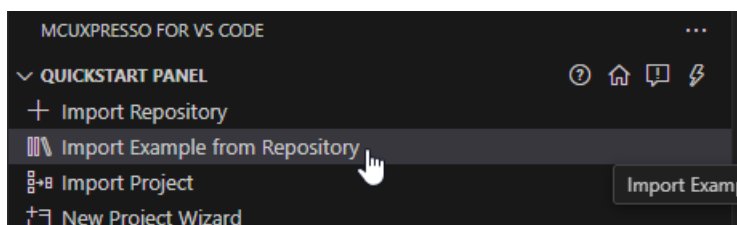


Note: You can import the SDK in several ways. Refer to [MCUXpresso for VS Code Wiki](#) for details.

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



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



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

Import Example from Repository

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

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

Board: FRDM-MCXC444 | ▾

 FRDM-MCXC444

Template: demo_apps/hello_world | ▾

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

App type: Freestanding application | ▾

Name: frdmxc444_hello_world

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

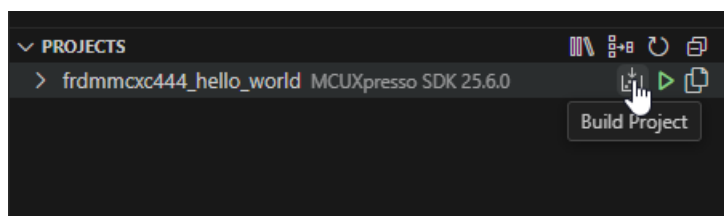
Note: Path doesn't exist. Folder(s) will be created.

☐ Open readme file after project is imported

[Import](#)

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

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

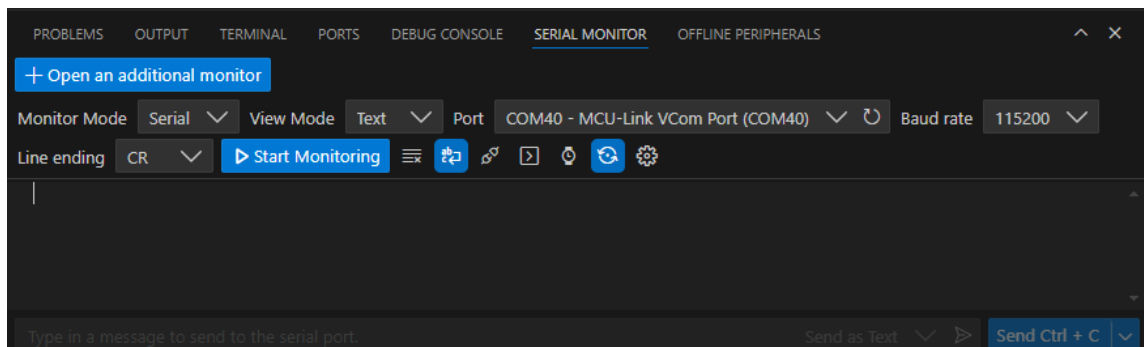
```

[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.

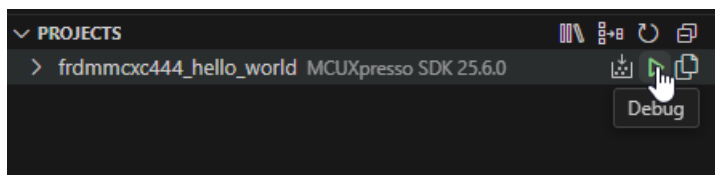
```

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

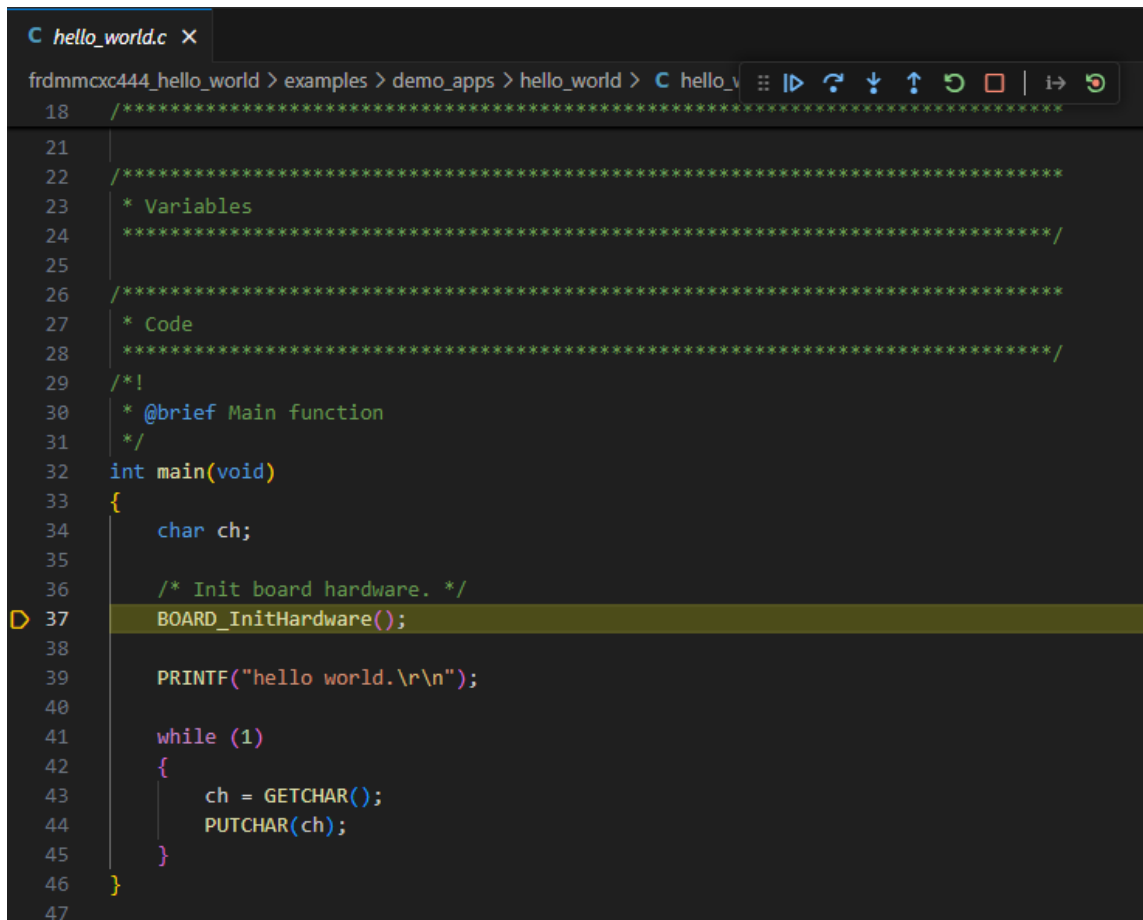
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



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.

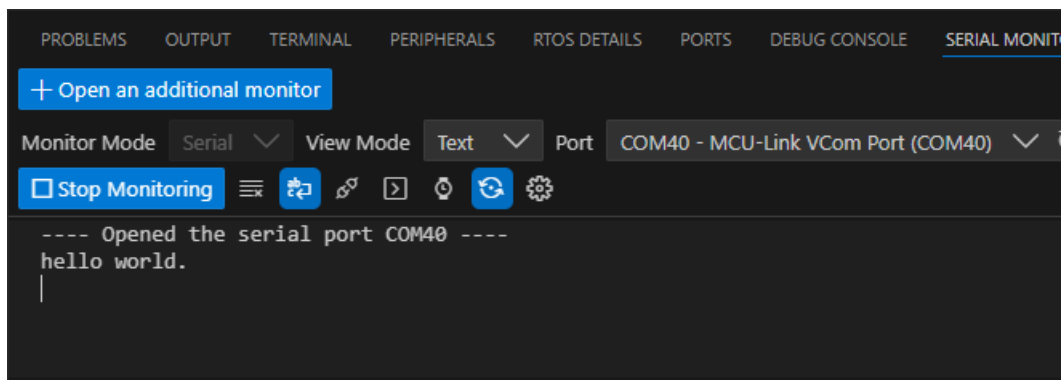


```

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)

```

↪ evkbnimxrt1060]
INFO: [ 4]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm4]
INFO: [ 5]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1170 -Dcore_id=cm7]
INFO: [ 6]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt1060]
INFO: [ 7]west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b_
↪ evkbnimxrt7ulp]
...

```

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 evkbnimxrt1170 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 evkbnimxrt1170 --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

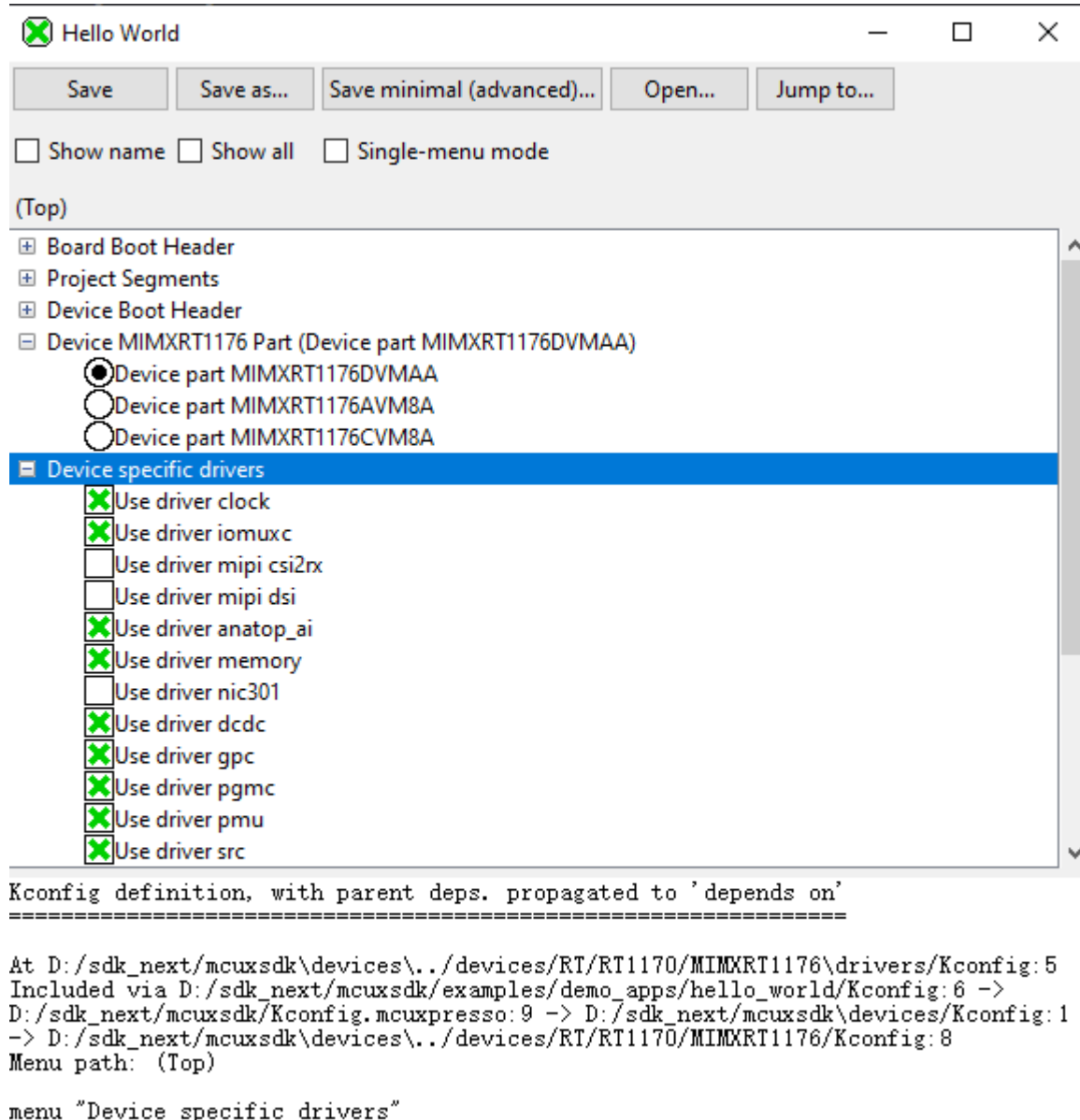
```
west build -b evkbnimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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:

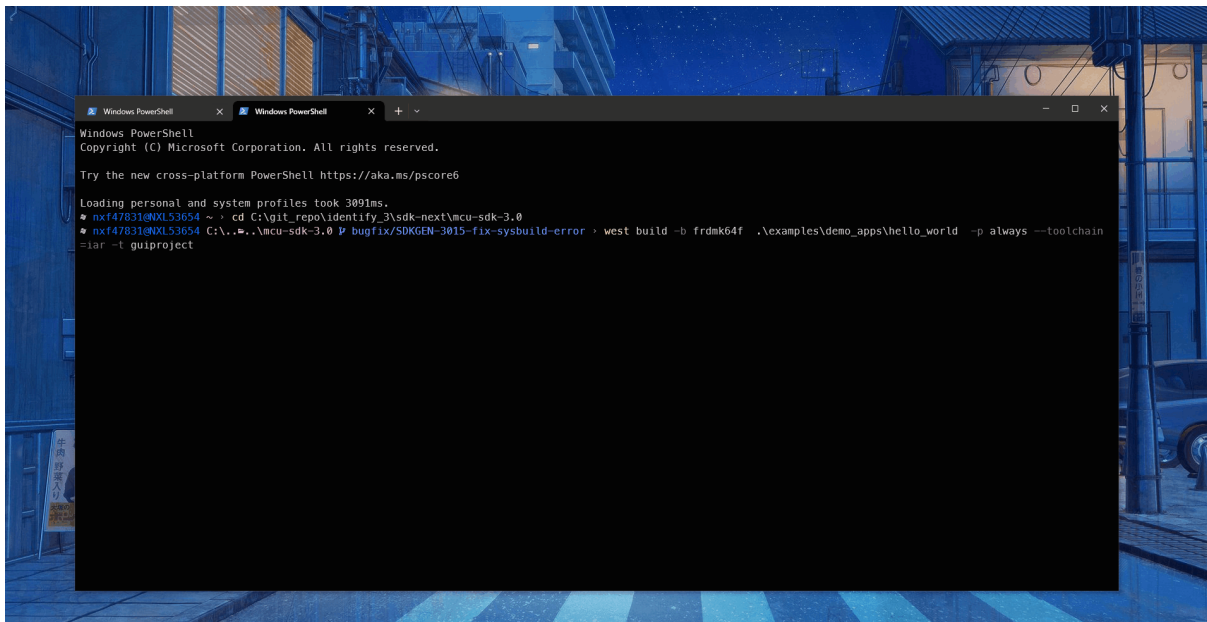
```
west debug -r linkserver
```

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.

Develop- ment boards	MCU devices
FRDM-MCXN236	MCXN235VDF, MCXN235VKL, MCXN235VNL, MCXN235VPB, MCXN236VDF , MCXN236VKL, MCXN236VNL, MCXN236VPB

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.

NXP IoT Agent NXP IoT Agent

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

Metering Metering Library

mbedTLS mbedtls SSL/TLS library v3.x

mbedTLS mbedtls SSL/TLS library v2.x

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.

Cannot add SDK components into FreeRTOS projects

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

Bluetooth LE link layer known issues/limitations

The known issues/limitations of the Bluetooth LE link layer include:

- Only one connection is supported.
- Only one advertising set is supported.
- Only one periodic advertising is supported.

FRDM-MCXN236 board audio rework issue

If a user needs to run audio cases, he needs to manually rework audio codec and jack.

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.

Max Blob Size limitation on N11

The Maximum blob size is limited to 2783 bytes instead of 8K bytes for el2go blobs.

Examples: el2go_blob_test_s, el2go_blob_test_ns

Affected toolchains: All

1.5 ChangeLog

1.5.1 MCUXpresso SDK Changelog

Board Support Files

board

[25.06.00]

- Initial version

clock_config

[25.06.00]

- Initial version

pin_mux

[25.06.00]

- Initial version
-

CACHE 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 `LP-CAC_CTRL[CLR_LPCAC]` should be set not clear.

[2.1.0]

- Improvements
 - Supported more features, such as write buffer control, 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**[2.0.0]**

- initial version.
-

MCX_CMC**[2.5.0]**

- New features
 - Update MCX_CMC driver to be compatible with new platforms.

[2.4.0]

- New features
 - Update MCX_CMC driver to be compatible with new platforms.

[2.3.0]

- Improvements
 - Added macros to support some device have BSR_SCR bit field.

[2.2.3]

- Improvements
 - Clear SCB SCR[SLEEPDEEP] bitfield after wakeup.

[2.2.2]

- Improvements
 - Fixed the violation of MISRA C-2012 rules.

[2.2.1]

- Improvements
 - Updated _cmc_system_reset_interrupt_enable, _cmc_system_reset_interrupt_flag and _cmc_system_reset_sources to support new added bit field.
- Bug Fixes
 - Fixed issue in CMC_PowerOffSRAMAllMode() and CMC_PowerOffSRAMLowPowerOnly() which overwrite reserved bit fields.

[2.2.0]

- Improvements
 - Added feature macro “FSL_FEATURE_MCX_CMC_HAS_NO_FLASHCR_WAKE” to support some devices where FLASHCR[WAKE] is reserved.

[2.1.0]

- Improvements
 - Added macros to support some devices(such as MCXA family) that only support one power domain.

[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.
-

EDMA

[2.10.7]

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

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

- Improvements
 - Applied ERRATA 51327.

[2.5.1]

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

[2.5.0]

- Improvements
 - Added feature FSL_FEATURE_EDMA_HAS_NO_SBR_ATTR_BIT to separate DMA without ATTR bitfield.
 - Added api EDMA_GetChannelSystemBusInformation to gets the channel identification and attribute information on the system bus interface.
- Bug Fixes
 - Fixed the ESG bit not set in scatter gather mode issue.

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

EDMA_SOC

[2.0.0]

- Initial version.
-

EIM

[2.0.4]

- Improvements
 - Change `eim_memory_channel_t` to `uint32_t`.
 - Support channel number 16-23.

[2.0.3]

- Bug Fixes
 - Fixed `s_eimResets` variable declaration issue.

[2.0.2]

- Improvements
 - Reset the peripheral during initialization, otherwise some platforms may not work.
- Bug Fixes
 - Fixed incorrect register access in `EIM_GetDataBitMask()`.

[2.0.1]

- Improvements
 - Update driver to support fewer channel.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.3.

[2.0.0]

- Initial version.
-

ERM

[2.0.4]

- Improvements
 - Change `erm_memory_channel_t` to `uint32_t`.
 - Support channel number 16-23.

[2.0.3]

- Bug Fixes
 - Fixed `s_ermResets` variable declaration issue.

[2.0.2]

- Improvements
 - Reset the peripheral during initialization, otherwise some platforms may not work.
- Bug Fixes
 - Only keep bit 0-3 of ERM_GetInterruptStatus() as it may get other channel results in higher bits.

[2.0.1]

- Improvements
 - Update driver to support fewer channel.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.3.

[2.0.0]

- Initial version.
-

EVTG

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - MISRA C-2012 issue fixed: rule 10.7,15.7,10.1,10.4,10.8,10.3,16.1,16.3 .

[2.0.0]

- Initial version.
-

EWM

[2.0.4]

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the hard fault in EWM_Deinit.

[2.0.0]

- Initial version.
-

FLEXCAN**[2.14.5]**

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

[2.14.4]

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

[2.14.3]

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

[2.14.2]

- Improvements
 - Add Coverage Justification for uncovered code.
 - Adjust API FLEXCAN__TransferAbortReceive order.
 - Update FLEXCAN__Enable to enter Freeze Mode first when enter Disable mode on some platform.
 - Added while loop timeout for following API:
 - * FLEXCAN__EnterFreezeMode
 - * FLEXCAN__ExitFreezeMode
 - * FLEXCAN__Enable
 - * FLEXCAN__Reset
 - * FLEXCAN__TransferSendBlocking
 - * FLEXCAN__TransferReceiveBlocking
 - * FLEXCAN__TransferFDSendBlocking
 - * FLEXCAN__TransferFDReceiveBlocking
 - * FLEXCAN__TransferReceiveFifoBlocking
 - * FLEXCAN__TransferReceiveEnhancedFifoBlocking
- Bug Fixes
 - Remove remote frame feature in CANFD mode because there is no remote frame in the CANFD format.
 - Remove legacy Rx FIFO disabled branch in FLEXCAN__SubHandlerForLegacyRxFIFO and FLEXCAN__SubHandlerForDataTransferred.

[2.14.1]

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

[2.14.0]

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

[2.13.1]

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

[2.13.0]

- Improvements
 - Support payload endianness selection feature.

[2.12.0]

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

[2.11.8]

- Improvements
 - Synchronize flexcan driver update on s32z platform.

[2.11.7]

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

[2.11.6]

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

[2.11.5]

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

[2.11.4]

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

[2.11.3]

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

[2.11.2]

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

[2.11.1]

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

[2.11.0]

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

[2.10.1]

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

[2.10.0]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.7]

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

[2.8.6]

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

[2.8.5]

- Improvements
 - Make driver aarch64 compatible.

[2.8.4]

- Bug Fixes
 - Fixed FlexCan_Errata_6032 to disable all interrupts.

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

- Improvements
 - Update config configuration.
 - * Added enablePretendedNetworking member to support enable/disable Pretended Networking feature.
 - * Added enableTransceiverDelayMeasure member to support enable/disable Transceiver Delay Measurement Pretended feature.
 - * Added bitRate/bitRateFD member to work as baudRate/baudRateFD member union.
 - Rename all “baud” in code or comments to “bit” to align with the CAN spec.
 - Added Pretended Networking mode related APIs.
 - * FLEXCAN_SetPNConfig
 - * FLEXCAN_GetPNMatchCount
 - * FLEXCAN_ReadPNWakeUpMB
 - Added support for Enhanced Rx FIFO.
 - Removed independent memory error interrupt/status APIs and put all interrupt/status control operation into FLEXCAN_EnableInterrupts/FLEXCAN_DisableInterrupts and FLEXCAN_GetStatusFlags/FLEXCAN_ClearStatusFlags APIs.
 - Update improved timing APIs to make it calculate improved timing according to CiA doc recommended.
 - * FLEXCAN_CalculateImprovedTimingValues.
 - * FLEXCAN_FDCalculateImprovedTimingValues.
 - Update FLEXCAN_SetBitRate/FLEXCAN_SetFDBitRate to added the use of enhanced timing registers.

[2.6.2]

- Improvements
 - Add CANFD frame data length enumeration.

[2.6.1]

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

[2.6.0]

- Improvements
 - Enable CANFD ISO mode in FLEXCAN_FDInit API.
 - Enable the transceiver delay compensation feature when enable FD operation and set bitrate switch.
 - Implementation memory error control in FLEXCAN_Init API.
 - Improve FLEXCAN_FDCalculateImprovedTimingValues API to get same value for FPRESDIV and PRESDIV.
 - Added memory error configuration for user.
 - * enableMemoryErrorControl
 - * enableNonCorrectableErrorEnterFreeze
 - Added memory error related APIs.
 - * FLEXCAN_GetMemoryErrorReportStatus
 - * FLEXCAN_GetMemoryErrorStatusFlags
 - * FLEXCAN_ClearMemoryErrorStatusFlags
 - * FLEXCAN_EnableMemoryErrorInterrupts
 - * FLEXCAN_DisableMemoryErrorInterrupts
- Bug Fixes
 - Fixed the issue of sent duff CAN frame after call FLEXCAN_FDInit() API.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Bug Fixes
 - MISRA C-2012 issue check.
 - * Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-11.8, rule-12.2, rule-13.4, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-16.4, rule-17.3, rule-5.8, rule-8.3, rule-8.5.
 - Fixed the issue that API FLEXCAN_SetFDRxMbConfig lacks inactive message buff.
 - Fixed the issue of Pa082 warning.
 - Fixed the issue of dead lock in the function of interruption handler.
 - Fixed the issue of Legacy Rx Fifo EDMA transfer data fail in evkmimxrt1060 and evk-mimxrt1064.
 - Fixed the issue of setting CANFD Bit Rate Switch.
 - Fixed the issue of operating unknown pointer risk.
 - * when used the pointer “handle->mbFrameBuf[mbIdx]” to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
 - Added assert to check current CAN clock source affected by other clock gates in current device.
 - * In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).
- Improvements
 - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
 - Optimized the function of interruption handler.
 - Added two APIs for FLEXCAN EDMA driver.
 - * FLEXCAN_PrepareTransfConfiguration
 - * FLEXCAN_StartTransferDatafromRxFIFO
 - Added new API for FLEXCAN driver.
 - * FLEXCAN_GetTimeStamp
 - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
 - * FLEXCAN_EnterFreezeMode
 - * FLEXCAN_ExitFreezeMode
 - Added new configuration for user.
 - * disableSelfReception
 - * enableListenOnlyMode
 - Renamed the two clock source enum macros based on CLKSRC bit field value directly.

- * The CLKSRC bit value had no property about Oscillator or Peripheral type in lots of devices, it acted as two different clock input source only, but the legacy enum macros name contained such property, that misled user to select incorrect CAN clock source.
- Created two new enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrc0
 - * kFLEXCAN_ClkSrc1
- Deprecated two legacy enum macros for the FLEXCAN driver.
 - * kFLEXCAN_ClkSrcOsc
 - * kFLEXCAN_ClkSrcPeri
- Changed the process flow for Remote request frame response..
 - * Created a new enum macro for the FLEXCAN driver.
 - kStatus_FLEXCAN_RxRemote
- Changed the process flow for kFLEXCAN_StateRxRemote state in the interrupt handler.
 - * Should the TX frame not register to the pointer of frame handle, interrupt handler would not be able to read the remote response frame from the mail box to ram, so user should read the frame by manual from mail box after a complete remote frame transfer.

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

FLEXCAN_EDMA

[2.12.1]

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

[2.12.0]

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

[2.11.7]

- Refer FLEXCAN driver change log 2.7.0 to 2.11.7
-

FLEXIO

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
 - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

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

[2.1.0]

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

[2.0.4]

- Bug Fixes
 - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.2]

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

[2.0.1]

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

FLEXIO_I2C**[2.6.2]**

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

[2.6.1]

- Bug Fixes
 - Fixed coverity issues

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

- New Features
 - Added timeout mechanism when waiting certain state in transfer API.
 - Added an API for checking bus pin status.
- Bug Fixes
 - Fixed COVERITY issue of useless call in `FLEXIO_I2C_MasterTransferRunStateMachine`.
 - Fixed MISRA issues
 - * Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

- Added codes in FLEXIO_I2C_MasterTransferCreateHandle to clear pending NVIC IRQ, disable internal IRQs before enabling NVIC IRQ.
- Modified code so that during master's nonblocking transfer the start and slave address are sent after interrupts being enabled, in order to avoid potential issue of sending the start and slave address twice.

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

- Bug Fixes
 - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
 - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S_EDMA**[2.1.9]**

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

[2.1.8]

- Improvements
 - Applied EDMA ERRATA 51327.
-

FLEXIO_MCU_LCD

[2.3.0]

- New Features
 - Supported passing an extra user defined parameter to GPIO functions to control the CS/RS/RDWR pin signal.

[2.2.0]

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

[2.1.0]

- New Features
 - Supported transmit only data without command.

[2.0.8]

- Bug Fixes
 - Fixed bug that FLEXIO_MCULCD_Init return kStatus_Success even with invalid parameter.
 - Fixed glitch on WR, that when initially configure the timer pin as output, or change the pin back to disabled, the pin may be driven low causing glitch on bus. Configure the pin as bidirection output first then perform a subsequent write to change to output or disabled to avoid the issue.

[2.0.6]

- Bug Fixes
 - Fixed MISRA 10.4 issues when FLEXIO_MCULCD_DATA_BUS_WIDTH defined as signed value.

[2.0.5]

- Improvements
 - Changed FLEXIO_MCULCD_WriteDataArrayBlocking's data parameter to const type.

[2.0.4]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 14.4, 17.7.

[2.0.2]

- Improvements
 - Unified component full name to FLEXIO_MCU_LCD (EDMA) driver.

[2.0.1]

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

[2.0.0]

- Initial version.
-

FLEXIO_MCU_LCD_EDMA**[2.0.6]**

- New Features
 - Supported passing an extra user defined parameter to GPIO functions to control the RDWR pin signal.

[2.0.5]

- New Features
 - Supported transmit only data without command.

[2.0.4]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 14.4, 17.7.

[2.0.2]

- Improvements
 - Unified component full name to FLEXIO_MCU_LCD (EDMA) driver.

[2.0.1]

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

[2.0.0]

- Initial version.
-

FLEXIO_MCU_LCD_SMARTDMA

[2.0.6]

- Other Changes
 - Add more MCXA devices support.

[2.0.5]

- Other Changes
 - Supported the MCXA platforms.
- New Features
 - Supported passing an extra user defined parameter to GPIO functions to control the RDWR pin signal.

[2.0.4]

- New Features
 - Supported the platforms which use FlexIO SHIFTER DMA to trigger SmartDMA, such as MCXN235, MCXN236.

[2.0.3]

- New Features
 - Supported transmit only data without command.

[2.0.2]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 14.4, 17.7.

[2.0.1]

- Other Changes
 - Update driver implementation due to SMARTDMA driver update.

[2.0.0]

- Initial version.
-

FLEXIO_SPI**[2.4.3]**

- Improvements
 - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

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

[2.4.1]

- Bug Fixes
 - Fixed coverity issues

[2.4.0]

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

[2.3.5]

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

[2.3.4]

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

[2.3.3]

- Bugfixes
 - Fixed cs-continuous mode.

[2.3.2]

- Improvements
 - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_UART**[2.6.4]**

- Improvements
 - Make UART_RETRY_TIMES configurable by CONFIG_UART_RETRY_TIMES.

[2.6.3]

- Bug Fixes
 - Fixed coverity issues

[2.6.2]

- Bug Fixes
 - Fixed coverity issues

[2.6.1]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

- Improvements
 - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_UART_EDMA

[2.3.1]

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

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0
-

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.
-

GDET

[2.1.0]

- Update for multiple instances
- Fix bug in isolation off API
- Add enable and disable APIs

[2.0.1]

- Fix MISRA in GDET_ReconfigureVoltageMode().

[2.0.0]

- Initial version.
-

GPIO

[2.8.2]

- Bug Fixes
 - Fixed COVERITY issue that GPIO_GetInstance could return clock array overflow values due to GPIO base and clock being out of sync.

[2.8.1]

- Bug Fixes
 - Fixed CERT INT31-C issues.

[2.8.0]

- Improvements
 - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.

[2.8.0]

- Improvements
 - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.
 - Remove support for API GPIO_GetPinsDMAResetFlags with GPIO_ISFR_COUNT <= 1.

[2.7.3]

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

[2.7.2]

- New Features
 - Support devices without PORT module.

[2.7.1]

- Bug Fixes
 - Fixed MISRA C-2012 rule 10.4 issues in GPIO_GpioGetInterruptChannelFlags() function and GPIO_GpioClearInterruptChannelFlags() function.

[2.7.0]

- New Features
 - Added API to support Interrupt select (IRQS) bitfield.

[2.6.0]

- New Features
 - Added API to get GPIO version information.
 - Added API to control a pin for general purpose input.
 - Added some APIs to control pin in secure and privilege status.

[2.5.3]

- Bug Fixes
 - Correct the feature macro typo: FSL_FEATURE_GPIO_HAS_NO_INDEP_OUTPUT_CONTORL.

[2.5.2]

- Improvements
 - Improved GPIO_PortSet/GPIO_PortClear/GPIO_PortToggle functions to support devices without Set/Clear/Toggle registers.

[2.5.1]

- Bug Fixes
 - Fixed wrong macro definition.
 - Fixed MISRA C-2012 rule issues in the FGPIO_CheckAttributeBytes() function.
 - Defined the new macro to separate the scene when the width of registers is different.
 - Removed some redundant macros.
- New Features
 - Added some APIs to get/clear the interrupt status flag when the port doesn't control pins' interrupt.

[2.4.1]

- Improvements
 - Improved GPIO_CheckAttributeBytes() function to support 8 bits width GACR register.

[2.4.0]

- Improvements
 - API interface added:
 - * New APIs were added to configure the GPIO interrupt clear settings.

[2.3.2]

- Bug Fixes
 - Fixed the issue for MISRA-2012 check.
 - * Fixed rule 3.1, 10.1, 8.6, 10.6, and 10.3.

[2.3.1]

- Improvements
 - Removed deprecated APIs.

[2.3.0]

- New Features
 - Updated the driver code to adapt the case of interrupt configurations in GPIO module. New APIs were added to configure the GPIO interrupt settings if the module has this feature on it.

[2.2.1]

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

[2.1.1]

- Improvements
 - API interface changes:
 - * Added an API for the check attribute bytes.

[2.1.0]

- Improvements
 - API interface changes:
 - * Added “pins” or “pin” to some APIs’ names.
 - * Renamed “_PinConfigure” to “GPIO_PinInit”.
-

I3C

[2.14.3]

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

[2.14.2]

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

[2.14.1]

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

[2.14.0]

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

[2.13.1]

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

[2.13.0]

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

[2.12.0]

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

[2.11.1]

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

[2.11.0]

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

[2.10.8]

- Improvements
 - Support more instances.

[2.10.7]

- Improvements
 - Fixed the potential compile warning.

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- New features
 - Added I3C extended IBI data support.

[2.9.0]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.4]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

- Bug Fixes
 - Fixed kI3C_SlaveMatchedFlag interrupt is not properly handled in I3C_SlaveTransferHandleIRQ when it comes together with interrupt kI3C_SlaveBusStartFlag.
 - Fixed the inaccurate I2C baudrate calculation in I3C_MasterSetBaudRate.
 - Added new API I3C_MasterGetIBIRules to get registered IBI rules.
 - Added new variable isReadTerm in struct i3c_master_handle for transfer state routine to check if MCTRL.RDTERM is configured for read transfer.
 - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
 - Fixed the slave maxWriteLength and maxReadLength does not be configured into SMAXLIMITS register issue.
 - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
 - Added isHotJoin in i3c_slave_config_t to request hot-join event during slave init.

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

I3C_EDMA**[2.2.12]**

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

[2.2.11]

- Improvements
 - Fixed Coverity CERT-C violations.

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Bug Fixes
 - Fixed MISRA issue rule 11.6.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
 - Deleted legacy IBI data request code.

[2.1.0]

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

[2.0.1]

- Bug Fixes
 - Fixed MISRA issue rule 9.1.

[2.0.0]

- Initial version.
-

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.
-

INTM**[2.1.0]**

- Replace macro FSL_FEATURE_INTM_MONITOR_COUNT to INTM_MON_COUNT.

[2.0.1]

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

[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.
-

LPCMP

[2.3.2]

- Improvements
 - Fixed LPCMP CERT-C issues.

[2.3.1]

- Improvements
 - Update LPCMP driver to be compatible with platforms that do not support LPCMP nano power mode selection.

[2.3.0]

- New Feature
 - Added some new features for platforms which support
 - * Plus input source selection.
 - * Minus input source selection.
 - * CMP to DAC link.
- Improvements
 - Removed some new features for platforms which doesn't support
 - * Functional clock source selection.
 - * DAC high power mode selection.
 - * Round Robin clock source selection.
 - * Round Robin trigger source selection.
 - * Round Robin channel sample numbers setting.
 - * Round Robin channel sample time threshold setting.
 - * Round Robin internal trigger configuration.

[2.2.0]

- Improvements
 - Change `FSL_FEATURE_LPCMP_HAS_NO_CCR0_CMP_STOP_EN` to `FSL_FEATURE_LPCMP_HAS_CCR0_CMP_STOP_EN`.

[2.1.3]

- New Feature
 - Added new macro to handle the case where some instances do not have the CCR0 CMP_STOP_EN bit field.

[2.1.2]

- New Feature
 - Add macros to be compatible with some platforms that do not have the CCR0 CMP_STOP_EN bitfield.

[2.1.1]

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

[2.1.0]

- New Features:
 - Supported round robin mode and window mode feature.

[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.
-

LPFLEXCOMM**[2.2.1]**

- Bug Fix
 - Fixed the bug that function LPFLEXCOMM_GetBaseAddr() type was uint32_t, but return NULL((void*)0).

[2.2.0]

- New Features
 - Added new API to get LPFLEXCOMM base address.

[2.1.2]

- Bug Fix
 - Fixed the bug that when the same instance is initialized multiple times, all previous states will be cleared and only the last state will be retained.

[2.1.1]

- New Features
 - Supported new platform that has more lpflexcomm instance.

[2.1.0]

- Improvements
 - Function LP_FLEXCOMM_SetPeriph changed from public to private,function LP_FLEXCOMM_Init no longer judges the periph.

[2.0.0]

- Initial version.
-

LPI2C

[2.2.6]

- Bug Fixes
 - Fixed coverity issues.

[2.2.5]

- Improvements
 - Added assert with target feature check in LPI2C_SlaveInit().
 - Added feature check to LPI2C_CommonIRQHandler().

[2.2.4]

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

[2.2.3]

- Bug Fixes
 - Fixed an issue that LP_FLEXCOMM_Deinit() is called incorrectly.

[2.2.2]

- Improvements
 - Fixed doxygen warning in LPI2C_SlaveTransferHandleIRQ.

[2.2.1]

- Bug Fixes
 - Added bus stop incase of bus stall in LPI2C_MasterTransferBlocking.

[2.2.0]

- Improvements
 - Support the normal LPI2C in LPFLEXCOMM driver.

[2.1.1]

- Improvements
 - Optimize slave ISR. When replying to ack/nack, first judge whether the user performs the reply in the APP.

[2.1.0]

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

[2.0.1]

- Improvements
 - Supported to initialize the flexcomm layer outside the peripheral driver initialization function.

[2.0.0]

- Initial version.
-

LPI2C_EDMA**[2.0.1]**

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

[2.0.0]

- Initial version.
-

LPSPi

[2.2.9]

- Bug Fixes
 - Fixed coverity issues.

[2.2.8]

- Bug Fixes
 - Fixed coverity issues.

[2.2.7]

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

[2.2.6]

- Bug Fixes
 - Fixed an issue that LP_FLEXCOMM_Deinit() is called incorrectly.

[2.2.5]

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

[2.2.4]

- Improvements
 - Fixed doxygen warning in LPSPi_SlaveTransferHandleIRQ.

[2.2.3]

- Bug Fixes
 - Disabled lpspi before LPSPi_MasterSetBaudRate incase of LPSPi opened.

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

- Fixed LPSPI_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

[2.0.0]

- Initial version.
-

LPSPI_EDMA**[2.1.4]**

- Improvements
 - Reduced DMA requests - transfer up to 4 FIFO words per DMA request

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

LPTMR

[2.2.1]

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

LPUART

[2.3.4]

- Bug Fixes
 - Fixed coverity issues.

[2.3.3]

- Bug Fixes
 - Fixed coverity issues.

[2.3.2]

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

[2.3.1]

- Bug Fix
 - Fixed MISRA C-2012 violations.

[2.3.0]

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

[2.2.4]

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

[2.2.3]

- Improvements
 - Added atomic in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.2.2]

- Improvements
 - Added comment on txExtendedTimeoutValue of lpuart_timeout_config_t.

[2.2.1]

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

[2.2.0]

- Improvements
 - Rename some enumeration variables to be consistent with lpuart driver.

[2.1.1]

- Improvements
 - Supported to initialize the flexcomm layer outside the peripheral driver initialization function.

[2.1.0]

- New Features
 - Supported new platform that does not have the feature of MODEM Control,MODEM Status,Receiver Extended Idle,Transmitter Extended Idle, Half Duplex Control,Timeout Control,Timeout Status and Timeout N.

[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.
-

NPX

[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.
-

OTP

[2.0.1]

- Bug Fixes
 - Fixed MISRA-C 2012 violations.

[2.0.0]

- Initial version.
-

PDM

[2.9.3]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
 - Added new features FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ/FSL_FEATURE_PDM_HAS_DC_OUT

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
 - Added software buffer queue for transactional API.

[2.0.1]

- Improvements
 - Improved HWVAD feature.

[2.0.0]

- Initial version.
-

PDM_EDMA

[2.6.5]

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

[2.6.4]

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

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.5.0
-

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.
-

PORT**[2.5.1]**

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.5.0]

- Bug Fixes
 - Correct the kPORT_MuxAsGpio for some platforms.

[2.4.1]

- Bug Fixes
 - Fixed the violations of MISRA C-2012 rules: 10.1, 10.8 and 14.4.

[2.4.0]

- New Features
 - Updated port_pin_config_t to support input buffer and input invert.

[2.3.0]

- New Features
 - Added new APIs for Electrical Fast Transient(EFT) detect.
 - Added new API to configure port voltage range.

[2.2.0]

- New Features
 - Added new api PORT_EnablePinDoubleDriveStrength.

[2.1.1]

- Bug Fixes
 - Fixed the violations of MISRA C-2012 rules: 10.1, 10.4–11.3–11.8, 14.4.

[2.1.0]

- New Features
 - Updated the driver code to adapt the case of the interrupt configurations in GPIO module. Will move the pin configuration APIs to GPIO module.

[2.0.2]

- Other Changes
 - Added feature guard macros in the driver.

[2.0.1]

- Other Changes
 - Added “const” in function parameter.
 - Updated some enumeration variables’ names.
-

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_SetOutputToIdle to set pwm channel output to idle.
 - Added API PWM_GetPwmChannelState to get the pwm channel output duty cycle value.
 - Added API PWM_SetPwmForceOutputToZero to set the pwm channel output to zero logic.
 - Added API PWM_SetChannelOutput to set the pwm channel output state.
 - Added API PWM_SetClockMode to set the value of the clock prescaler.

- Added API PWM_SetupPwmPhaseShift to set PWM which a special phase shift and 50% duty cycle.
- Added API PWM_SetVALxValue/PWM_GetVALxValue to set/get PWM VALs registers values directly.

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

- Improvements
 - Moved the configuration of fault input filter into a new API to avoid be initialized multiple times.
- Bug Fixes
 - MISRA C-2012 issue fixed.

- * Fix rules, containing: rule-10.2, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-14.4, rule-16.4.

[2.0.1]

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

[2.0.0]

- Initial version.
-

QDC

[2.0.0]

- Initial version. Rename from driver ENC.
-

RESET

[2.4.0]

- Improvements
 - Add RESET_ReleasePeripheralReset API.

[2.0.0]

- Initial version.
-

SAI

[2.4.10]

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

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

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

[2.4.7]

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

[2.4.6]

- Bug Fixes
 - Fixed the IAR build warning.

[2.4.5]

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

[2.4.4]

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

[2.4.3]

- Bug Fixes
 - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
 - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
 - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

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

[2.3.6]

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

[2.3.5]

- Improvements
 - Make driver to be aarch64 compatible.

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

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

[2.0.0]

- Initial version.
-

SAI_EDMA

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
 - Removed deprecated APIs.

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2
-

SMARTDMA**[2.14.0]**

- Improvements
 - Refine the device specific firmware selection method, use device Kconfig symbol to select.

[2.13.3]

- Bug Fixes
 - Fix RT500 FlexIO QSPI MDK build warning.

[2.13.2]

- Other Changes
 - Add more MCXN devices support.

[2.13.1]

- Other Changes
 - Add more MCXA devices support.

[2.13.0]

- New Features
 - Added MCXA keyscan firmware.
 - Added functions to control CTRL[EXF].

[2.12.0]

- New Features
 - Supported MCXA mculcd and camera functions.

[2.11.0]

- Improvements
 - Make the RT500 QSPI firmware can work with other display firmware in the same project.

[2.10.0]

- New Features
 - Added new camera APIs for MCXN SoCs to support more resolutions.

[2.9.1]

- New Features
 - Supported MCXN235, MCXN236.

[2.9.0]

- New Features
 - Supported MCXN camera functions.
 - Supported user to select individual firmware for MIPI or FLEXIO alone, or both.
 - Added new API of enabling DMA from FlexIO to a Buffer.
 - Added new APIs of setting MIPI-DSI to enter and exit ultra low power state.

[2.8.0]

- New Features
 - Supported converting the pixel data from RGB565 to RGB888.
 - Supported function to turn off certain pixel in a checker board pattern.

[2.7.0]

- New Features
 - Supported data transfer in 2-dimensional way.
 - Supported data transfer in XRGB8888 format and rotate 180 degree.
 - Supported to fill data in whenever there is room in MIPI controller's FIFO rather than using the tx FIFO in double buffered way.

[2.6.3]

- Bug Fixes
 - Fixed EZH_MIPIDSI_RGB565_DMA, EZH_MIPIDSI_RGB888_DMA, EZH_MIPIDSI_ARGB888toRGB888_DMA issues that don't support some length.

[2.6.2]

- Bug Fixes
 - Fixed MISRA C-2012 issues: 8.4, 11.6, 17.7.

[2.6.1]

- Improvements
 - Optimized MIPI DSI APIs performance.

[2.6.0]

- Improvements
 - Optimized MIPI DSI APIs performance.
- New Features
 - Added new APIs to send MIPI DSI frame with 180 degree rotation.

[2.5.0]

- Improvements
 - Supported swap or don't swap the pixel byte before written to MIPI DSI FIFO.
 - Updated MIPI DSI firmware, make sure data has been sent out before calling callback function.

[2.4.0]

- Improvements
 - Added new APIs for MIPI DSI kSMARTDMA_MIPI_XRGB2RGB_DMA.

[2.3.0]

- Improvements
 - Added new APIs for FlexIO one SHIFTBUF, kSMARTDMA_FlexIO_DMA_ONELANE.
- Bug Fixes
 - Fixed kSMARTDMA_MIPI_RGB565_DMA color bias issue.

[2.2.0]

- Improvements
 - Added new APIs for MIPI DSI, kSMARTDMA_MIPI_RGB565_DMA and kSMARTDMA_MIPI_RGB888_DMA.
 - Supported install firmware and callback function dynamically.

[2.1.0]

- Improvements
 - Removed test APIs, including kSMARTDMA_LightOn, kSMARTDMA_LightOff, kSMARTDMA_Notify, and kSMARTDMA_Test.
 - Added new APIs, including kSMARTDMA_FlexIO_DMA_Reverse, kSMARTDMA_FlexIO_DMA_ARGB2RGB, kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap, and kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap_Reverse.

[2.0.0]

- Initial version.
-

MCX_SPC

[2.10.0]

- New Features
 - Add feature macro to be compatible with some platforms where SPC does not support:
 1. Has no SC register SPC_LP_REQ bitfield.
 2. Has no SC register SPC_LP_MODE bitfield.
 3. Has no PD_STATUS register.
 4. Has no SRAMCTL register.

[2.9.1]

- Improvements
 - There is no need to ensure that the bandgap is enabled before setting the active mode core LDO regulator drive strength and voltage level, because the hardware will automatically turn on the bandgap function if it is required by the hardware when configuring the LDO.

[2.9.0]

- New Features
 - Add feature macro to be compatible with some platforms where SPC does not support overdrive voltage.

[2.8.1]

- Improvements
 - Fixed cert_int31_c_violation of SPC_GetVoltageDetectStatusFlag(), SPC_GetExternalDomainsStatus() and SPC_SetDCDCBurstConfig.

[2.8.0]

- New Features
 - Updated MCX_SPC driver for compatibility with SoCs without PD_STATUS[PWR_REQ_STATUS].

[2.7.0]

- New Features
 - Added new function to unretain RAM array.

[2.6.0]

- Bug Fixes
 - The enumeration kSPC_DeepPowerDownWithSysClockOff should be 0x8U.
- New Features
 - Added functions to get the last low-power mode that the power domain requested.

[2.5.0]

- Improvements
 - Updated SPC_SetLowPowerModeCoreLDORegulatorConfig() with adding check that LP_CFG[CORELDO_VDD_LVL] only be updated when CORELDO is in normal driver strength in active mode.
 - Updated SPC_SetLowPowerModeRegulatorsConfig() with adding check the if DCDC voltage level set as retention mode, CORELDO low voltage detection must be disabled.
 - Updated spc_analog_module_control with adding Opamp3 support.
- New Features
 - Added functions to mask/unmask voltage detections in active mode.

[2.4.2]

- Improvements
 - Fixed the violation of MISRA C-2012 rules.

[2.4.1]

- Improvements
 - Fixed the violation of MISRA C-2012 rules.

[2.4.0]

- Improvements
 - Refined APIs to set regulators, the input parameters will be check firstly before setting register.
 - Improved APIs' document.
 - Removed software check for DCDC's settings since there is not hardware restrictions for DCDC.
 - Added new APIs to check if glitch detector is enabled in active/low power mode.
 - Added new APIs for DCDC burst feature, make it more flexible to use.
 - Added new API to enable/disable DCDC bleed resistor.
 - Set functions of VD_SYS_CFG[LVSEL] configuration as deprecated, since this bit filed is reserved for all devices.

- Updated details of `spc_core_ldo_voltage_level_t` to align with description of latest RM, added comments to reminder users that the retention voltage is reserved in some devices.

[2.3.0]

- New Features
 - Updated glitch detect API to support devices which do not have aGDET

[2.2.1]

- Bug Fixes
 - Fixed an issue of `SPC_SetActiveModeRegulatorsConfig()` which will cause LVD in case of setting DCDC and CORE LDO into higher voltage level.

[2.2.0]

- New Features
 - Added some macros to support some devices(such as MCXA family) do not equipped DCDC, SYS_LDO and so on.
 - Added new function `SPC_EnableSRAMLdoLowPowerModeIREF()`, `SPC_TrimSRAMLdoRefVoltage()`, `SPC_EnableSRAMLdo()` to support some devices(such as MCXA family) that support control of SRAM_LDO.
 - Fixed violation of MISRA C-2012 rule 17.7.
 - Fixed an issue in `SPC_SetLowPowerModeRegulatorsConfig()` function that set ACTIVE_CFG register by mistake.

[2.1.0]

- Improvements
 - Added new functions to set regulators' voltage level and drive strength individually.
 - Updated `SPC_SetActiveModeRegulatorsConfig()` and `SPC_SetLowPowerModeRegulatorsConfig()` based on new added functions.

[2.0.1]

- Bug Fixes
 - Fixed a bug that external burst not working properly.
 - Fixed a bug that is SPC has VDD_DS the voltage is not configured correctly.

[2.0.0]

- Initial version.
-

SYSPM

[2.3.1]

- Improvements
 - Add timeout for while loop.

[2.3.0]

- New Features
 - Supported instruction counter.

[2.2.0]

- New Features
 - Supported platforms which only have one monitor.
 - Supported platforms whose instruction counter can't be cleared.
 - Supported platforms whose counter can't be disabled.

[2.1.0]

- New Features
 - Added new APIs SYSPM_Init and SYSPM_Deinit.

[2.0.0]

- Initial version.
-

TDET

[2.3.0]

- Added enum for TIF10.

[2.2.0]

- Added support for chips without active tamper pins.

[2.1.1]

- Added clearing SR_TAF and SR_DTF into TDET_Init().
 - Fix typo in kTDET_ClockType64Hz comment.

[2.1.0]

- Added setting of disabling prescaler on tamper event into TDET_SetConfig() and TDET_GetDefaultConfig functions.

[2.0.0]

- Initial version.
-

TRDC

[2.2.2]

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

[2.2.1]

- Bug Fixes:
 - Fix MISRA violations.

[2.2.0]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

TRDC_SOC

[2.0.0]

- Initial version.
-

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.
-

MCX_VBAT**[2.5.0]**

- New features
 - Update MCX_VBAT driver to be compatible with new platforms.

[2.4.0]

- New features
 - Update MCX_VBAT driver to be compatible with new platforms.

[2.3.1]

- Bug Fixes
 - Fixed violations of MISRA C-2012 low impact rules.

[2.3.0]

- Improvements
 - OSCCTLA[FINE_AMP_GAIN] is reserved, added new macro to support this change.
 - Defined VBAT_LDORAMC_RET in case of this macro is not defined in header file.
- Bug Fixes
 - Fixed violations of MISRA C-2012 rule 10.8.

[2.2.0]

- Improvements
 - Added macros to support some devices(such as MCXA family) that do not support OSC control, LDO control, bandgap timer, and tamper.

[2.1.0]

- New features
 - Added functions to support tamper and clock monitor.

[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 type 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.
-

WUU

[2.4.1]

- Improvements
 - The semantics of kWUU_FilterActiveDSPD and kWUU_ExternalPinActiveDSPD are not clear enough, because on some platforms it's not 'DSPD' (deep sleep, power down) but PDDPD (power down, deep power down). We deprecate them and will use the more generic kWUU_FilterActiveLowLeakage and kWUU_ExternalPinActiveLowLeakage in the future.

[2.4.0]

- New Features
 - Added WUU_ClearExternalWakeupPinsConfig() to clear settings of PDC and PE register.

[2.3.0]

- New Features
 - Added WUU_ClearInternalWakeUpModulesConfig() to clear settings of DM and ME register.

[2.2.1]

- Bug Fixes
 - Fixed WUU_SetPinFilterConfig() unable to set edge detection of pin filter config.
 - Fixed wrong macro used in WUU_GetPinFilterFlag() function.

[2.2.0]

- New Features
 - Added the WUU_GetExternalWakeupPinFlag() and WUU_ClearExternalWakeupPinFlag() function .

[2.1.0]

- New Features
 - Added the `WUU_GetModuleInterruptFlag()` function to support the devices that equipped MF register.

[2.0.0]

- Initial version.
-

WWDT

[2.1.10]

- Bug Fixes
 - Check `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 WWWDWT 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.

[MCXN236](#)

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

MCXN236

2.1 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

2.2 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>
```

2.3 Clock Driver

```
enum _clock_ip_name
```

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

```
enumerator kCLOCK_IpInvalid
```

Invalid Ip Name.

```
enumerator kCLOCK_None
```

None clock gate.

```
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_Sram5
Clock gate name: Sram5.

enumerator kCLOCK_Sram6
Clock gate name: Sram6.

enumerator kCLOCK_Sram7
Clock gate name: Sram7.

enumerator kCLOCK_Fmu
Clock gate name: Fmu.

enumerator kCLOCK_Fmc
Clock gate name: Fmc.

enumerator kCLOCK_InputMux0
Clock gate name: InputMux0.

enumerator kCLOCK_InputMux
Clock gate name: InputMux0.

enumerator kCLOCK_Port0
Clock gate name: Port0.

enumerator kCLOCK_Port1
Clock gate name: Port1.

enumerator kCLOCK_Port2
Clock gate name: Port2.

enumerator kCLOCK_Port3
Clock gate name: Port3.

enumerator kCLOCK_Port4
Clock gate name: Port4.

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_Gpio4
Clock gate name: Gpio4.

enumerator kCLOCK_Pint
Clock gate name: Pint.

enumerator kCLOCK_Dma0
Clock gate name: Dma0.

enumerator kCLOCK_Crc0
Clock gate name: Crc.

enumerator kCLOCK_Wwdt0
Clock gate name: Wwdt0.

enumerator kCLOCK_Wwdt1
Clock gate name: Wwdt1.

enumerator kCLOCK_Mrt
Clock gate name: Mrt.

enumerator kCLOCK_OsTimer
Clock gate name: OsTimer.

enumerator kCLOCK_Sct
Clock gate name: Sct.

enumerator kCLOCK_Adc0
Clock gate name: Adc0.

enumerator kCLOCK_Adc1
Clock gate name: Adc1.

enumerator kCLOCK_Rtc0
Clock gate name: Rtc.

enumerator kCLOCK_Utick
Clock gate name: Utick.

enumerator kCLOCK_LPFlexComm0
Clock gate name: LPFlexComm0.

enumerator kCLOCK_LPFlexComm1
Clock gate name: LPFlexComm1.

enumerator kCLOCK_LPFlexComm2
Clock gate name: LPFlexComm2.

enumerator kCLOCK_LPFlexComm3
Clock gate name: LPFlexComm3.

enumerator kCLOCK_LPFlexComm4
Clock gate name: LPFlexComm4.

enumerator kCLOCK_LPFlexComm5
Clock gate name: LPFlexComm5.

enumerator kCLOCK_LPFlexComm6
Clock gate name: LPFlexComm6.

enumerator kCLOCK_LPFlexComm7
Clock gate name: LPFlexComm7.

enumerator kCLOCK_LPUart0
Clock gate name: LPUart0.

enumerator kCLOCK_LPUart1
Clock gate name: LPUart1.

enumerator kCLOCK_LPUart2
Clock gate name: LPUart2.

enumerator kCLOCK_LPUart3
Clock gate name: LPUart3.

enumerator kCLOCK_LPUart4
Clock gate name: LPUart4.

enumerator kCLOCK_LPUart5
Clock gate name: LPUart5.

enumerator kCLOCK_LPUart6
Clock gate name: LPUart6.

enumerator kCLOCK_LPUart7
Clock gate name: LPUart7.

enumerator kCLOCK_LPSpi0
Clock gate name: LPSpi0.

enumerator kCLOCK_LPSpi1
Clock gate name: LPSpi1.

enumerator kCLOCK_LPSpi2
Clock gate name: LPSpi2.

enumerator kCLOCK_LPSpi3
Clock gate name: LPSpi3.

enumerator kCLOCK_LPSpi4
Clock gate name: LPSpi4.

enumerator kCLOCK_LPSpi5
Clock gate name: LPSpi5.

enumerator kCLOCK_LPSpi6
Clock gate name: LPSpi6.

enumerator kCLOCK_LPSpi7
Clock gate name: LPSpi7.

enumerator kCLOCK_LPI2c0
Clock gate name: LPI2c0.

enumerator kCLOCK_LPI2c1
Clock gate name: LPI2c1.

enumerator kCLOCK_LPI2c2
Clock gate name: LPI2c2.

enumerator kCLOCK_LPI2c3
Clock gate name: LPI2c3.

enumerator kCLOCK_LPI2c4
Clock gate name: LPI2c4.

enumerator kCLOCK_LPI2c5
Clock gate name: LPI2c5.

enumerator kCLOCK_LPI2c6
Clock gate name: LPI2c6.

enumerator kCLOCK_LPI2c7
Clock gate name: LPI2c7.

enumerator kCLOCK_Micfil
Clock gate name: Micfil.

enumerator kCLOCK_Timer2
Clock gate name: Timer2.

enumerator kCLOCK_Usb0FsDcd
Clock gate name: Usb0FsDcd.

enumerator kCLOCK_Timer0
Clock gate name: Timer0.

enumerator kCLOCK_Timer1
Clock gate name: Timer1.

enumerator kCLOCK_PkcRam
Clock gate name: PkcRam.

enumerator kCLOCK_Smartdma
Clock gate name: SmartDma.

enumerator kCLOCK_Espi
Clock gate name: Espi.

enumerator kCLOCK_Dma1
Clock gate name: Dma1.

enumerator kCLOCK_Flexio
Clock gate name: Flexio.

enumerator kCLOCK_Sai0
Clock gate name: Sai0.

enumerator kCLOCK_Sai1
Clock gate name: Sai1.

enumerator kCLOCK_Tro
Clock gate name: Tro.

enumerator kCLOCK_Freqme
Clock gate name: Freqme.

enumerator kCLOCK_Trng
Clock gate name: Trng.

enumerator kCLOCK_Flexcan0
Clock gate name: Flexcan0.

enumerator kCLOCK_Flexcan1
Clock gate name: Flexcan1.

enumerator kCLOCK_UsbHs
Clock gate name: UsbHs.

enumerator kCLOCK_UsbHsPhy
Clock gate name: UsbHsPhy.

enumerator kCLOCK_Css
Clock gate name: Css.

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_Scg
Clock gate name: Scg.

enumerator kCLOCK_Gdet
Clock gate name: Gdet.

enumerator kCLOCK_Sm3
Clock gate name: Sm3.

enumerator kCLOCK_I3c0
Clock gate name: I3c0.

enumerator kCLOCK_I3c1
Clock gate name: I3c1.

enumerator kCLOCK_Qdc0
Clock gate name: Qdc0.

enumerator kCLOCK_Qdc1
Clock gate name: Qdc1.

enumerator kCLOCK_Pwm0
Clock gate name: Pwm0.

enumerator kCLOCK_Pwm1
Clock gate name: Pwm1.

enumerator kCLOCK_Evtg
Clock gate name: Evtg.

enumerator kCLOCK_Cmp2
Clock gate name: Cmp2.

enumerator kCLOCK_Vref
Clock gate name: Vref.

enumerator kCLOCK_Ewm
Clock gate name: Ewm.

enumerator kCLOCK_Ewm0
Clock gate name: Ewm.

enumerator kCLOCK_Eim
Clock gate name: Eim.

enumerator kCLOCK_Erm
Clock gate name: Erm.

enumerator kCLOCK_Intm
Clock gate name: Intrn.

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_SystickClk0

Systick clock0

enumerator kCLOCK_ClockOut

CLOCKOUT

enumerator kCLOCK_Fro12M

FRO12M

enumerator kCLOCK_Clk1M

CLK1M

enumerator kCLOCK_FroHf

FRO48/144

enumerator kCLOCK_Clk48M

CLK48M

enumerator kCLOCK_Clk144M

CLK144M

enumerator kCLOCK_Clk16K0

CLK16K[0]

enumerator kCLOCK_Clk16K1

CLK16K[1]

enumerator kCLOCK_Clk16K2

CLK16K[2]

enumerator kCLOCK_Clk16K3
CLK16K[3]

enumerator kCLOCK_ExtClk
External Clock

enumerator kCLOCK_Osc32K0
OSC32K[0]

enumerator kCLOCK_Osc32K1
OSC32K[1]

enumerator kCLOCK_Osc32K2
OSC32K[2]

enumerator kCLOCK_Osc32K3
OSC32K[3]

enumerator kCLOCK_Pll0Out
PLL0 Output

enumerator kCLOCK_Pll1Out
PLL1 Output

enumerator kCLOCK_UsbPllOut
USB PLL Output

enumerator kCLOCK_LpOsc
lp_osc

enum _clock_attach_id

The enumerator of clock attach Id.

Values:

enumerator kCLK_IN_to_MAIN_CLK
Attach clk_in to MAIN_CLK.

enumerator kFRO12M_to_MAIN_CLK
Attach FRO_12M to MAIN_CLK.

enumerator kFRO_HF_to_MAIN_CLK
Attach FRO_HF to MAIN_CLK.

enumerator kXTAL32K2_to_MAIN_CLK
Attach xtal32k[2] to MAIN_CLK.

enumerator kPLL0_to_MAIN_CLK
Attach PLL0 to MAIN_CLK.

enumerator kPLL1_to_MAIN_CLK
Attach PLL1 to MAIN_CLK.

enumerator kUSB_PLL_to_MAIN_CLK
Attach USB PLL to MAIN_CLK.

enumerator kNONE_to_MAIN_CLK
Attach NONE to MAIN_CLK.

enumerator kSYSTICK_DIV0_to_SYSTICK0
Attach SYSTICK_DIV0 to SYSTICK0.

enumerator kCLK_1M_to_SYSTICK0
Attach Clk 1 MHz to SYSTICK0.

enumerator kLPOSC_to_SYSTICK0
Attach LP Oscillator to SYSTICK0.

enumerator kNONE_to_SYSTICK0
Attach NONE to SYSTICK0.

enumerator kTRACE_DIV_to_TRACE
Attach TRACE_DIV to TRACE.

enumerator kCLK_1M_to_TRACE
Attach Clk 1 MHz to TRACE.

enumerator kLPOSC_to_TRACE
Attach LP Oscillator to TRACE.

enumerator kNONE_to_TRACE
Attach NONE to TRACE.

enumerator kCLK_1M_to_CTIMER0
Attach CLK_1M to CTIMER0.

enumerator kPLL0_to_CTIMER0
Attach PLL0 to CTIMER0.

enumerator kPLL1_CLK0_to_CTIMER0
Attach PLL1_clk0 to CTIMER0.

enumerator kFRO_HF_to_CTIMER0
Attach FRO_HF to CTIMER0.

enumerator kFRO12M_to_CTIMER0
Attach FRO 12MHz to CTIMER0.

enumerator kSAI0_MCLK_IN_to_CTIMER0
Attach SAI0 MCLK IN to CTIMER0.

enumerator kLPOSC_to_CTIMER0
Attach LP Oscillator to CTIMER0.

enumerator kSAI1_MCLK_IN_to_CTIMER0
Attach SAI1 MCLK IN to CTIMER0.

enumerator kSAI0_TX_BCLK_to_CTIMER0
Attach SAI0 TX_BCLK to CTIMER0.

enumerator kSAI0_RX_BCLK_to_CTIMER0
Attach SAI0 RX_BCLK to CTIMER0.

enumerator kSAI1_TX_BCLK_to_CTIMER0
Attach SAI1 TX_BCLK to CTIMER0.

enumerator kSAI1_RX_BCLK_to_CTIMER0
Attach SAI1 RX_BCLK to CTIMER0.

enumerator kNONE_to_CTIMER0
Attach NONE to CTIMER0.

enumerator kCLK_1M_to_CTIMER1
Attach CLK_1M to CTIMER1.

enumerator kPLL0_to_CTIMER1
Attach PLL0 to CTIMER1.

enumerator kPLL1_CLK0_to_CTIMER1
Attach PLL1_clk0 to CTIMER1.

enumerator kFRO_HF_to_CTIMER1
Attach FRO_HF to CTIMER1.

enumerator kFRO12M_to_CTIMER1
Attach FRO 12MHz to CTIMER1.

enumerator kSAI0_MCLK_IN_to_CTIMER1
Attach SAI0 MCLK IN to CTIMER1.

enumerator kLPOSC_to_CTIMER1
Attach LP Oscillator to CTIMER1.

enumerator kSAI1_MCLK_IN_to_CTIMER1
Attach SAI1 MCLK IN to CTIMER1.

enumerator kSAI0_TX_BCLK_to_CTIMER1
Attach SAI0 TX_BCLK to CTIMER1.

enumerator kSAI0_RX_BCLK_to_CTIMER1
Attach SAI0 RX_BCLK to CTIMER1.

enumerator kSAI1_TX_BCLK_to_CTIMER1
Attach SAI1 TX_BCLK to CTIMER1.

enumerator kSAI1_RX_BCLK_to_CTIMER1
Attach SAI1 RX_BCLK to CTIMER1.

enumerator kNONE_to_CTIMER1
Attach NONE to CTIMER1.

enumerator kCLK_1M_to_CTIMER2
Attach CLK_1M to CTIMER2.

enumerator kPLL0_to_CTIMER2
Attach PLL0 to CTIMER2.

enumerator kPLL1_CLK0_to_CTIMER2
Attach PLL1_clk0 to CTIMER2.

enumerator kFRO_HF_to_CTIMER2
Attach FRO_HF to CTIMER2.

enumerator kFRO12M_to_CTIMER2
Attach FRO 12MHz to CTIMER2.

enumerator kSAI0_MCLK_IN_to_CTIMER2
Attach SAI0 MCLK IN to CTIMER2.

enumerator kLPOSC_to_CTIMER2
Attach LP Oscillator to CTIMER2.

enumerator kSAI1_MCLK_IN_to_CTIMER2
Attach SAI1 MCLK IN to CTIMER2.

enumerator kSAI0_TX_BCLK_to_CTIMER2
Attach SAI0 TX_BCLK to CTIMER2.

enumerator kSAI0_RX_BCLK_to_CTIMER2

Attach SAI0 RX_BCLK to CTIMER2.

enumerator kSAI1_TX_BCLK_to_CTIMER2

Attach SAI1 TX_BCLK to CTIMER2.

enumerator kSAI1_RX_BCLK_to_CTIMER2

Attach SAI1 RX_BCLK to CTIMER2.

enumerator kNONE_to_CTIMER2

Attach NONE to CTIMER2.

enumerator kCLK_1M_to_CTIMER3

Attach CLK_1M to CTIMER3.

enumerator kPLL0_to_CTIMER3

Attach PLL0 to CTIMER3.

enumerator kPLL1_CLK0_to_CTIMER3

Attach PLL1_clk0 to CTIMER3.

enumerator kFRO_HF_to_CTIMER3

Attach FRO_HF to CTIMER3.

enumerator kFRO12M_to_CTIMER3

Attach FRO 12MHz to CTIMER3.

enumerator kSAI0_MCLK_IN_to_CTIMER3

Attach SAI0 MCLK IN to CTIMER3.

enumerator kLPOSC_to_CTIMER3

Attach LP Oscillator to CTIMER3.

enumerator kSAI1_MCLK_IN_to_CTIMER3

Attach SAI1 MCLK IN to CTIMER3.

enumerator kSAI0_TX_BCLK_to_CTIMER3

Attach SAI0 TX_BCLK to CTIMER3.

enumerator kSAI0_RX_BCLK_to_CTIMER3

Attach SAI0 RX_BCLK to CTIMER3.

enumerator kSAI1_TX_BCLK_to_CTIMER3

Attach SAI1 TX_BCLK to CTIMER3.

enumerator kSAI1_RX_BCLK_to_CTIMER3

Attach SAI1 RX_BCLK to CTIMER3.

enumerator kNONE_to_CTIMER3

Attach NONE to CTIMER3.

enumerator kCLK_1M_to_CTIMER4

Attach CLK_1M to CTIMER4.

enumerator kPLL0_to_CTIMER4

Attach PLL0 to CTIMER4.

enumerator kPLL1_CLK0_to_CTIMER4

Attach PLL1_clk0 to CTIMER4.

enumerator kFRO_HF_to_CTIMER4

Attach FRO_HF to CTIMER4.

enumerator kFRO12M_to_CTIMER4
Attach FRO 12MHz to CTIMER4.

enumerator kSAI0_MCLK_IN_to_CTIMER4
Attach SAI0 MCLK IN to CTIMER4.

enumerator kLPOSC_to_CTIMER4
Attach LP Oscillator to CTIMER4.

enumerator kSAI1_MCLK_IN_to_CTIMER4
Attach SAI1 MCLK IN to CTIMER4.

enumerator kSAI0_TX_BCLK_to_CTIMER4
Attach SAI0 TX_BCLK to CTIMER4.

enumerator kSAI0_RX_BCLK_to_CTIMER4
Attach SAI0 RX_BCLK to CTIMER4.

enumerator kSAI1_TX_BCLK_to_CTIMER4
Attach SAI1 TX_BCLK to CTIMER4.

enumerator kSAI1_RX_BCLK_to_CTIMER4
Attach SAI1 RX_BCLK 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 kCLK_IN_to_CLKOUT
Attach Clk_in to CLKOUT.

enumerator kFRO_HF_to_CLKOUT
Attach FRO_HF to CLKOUT.

enumerator kFRO12M_to_CLKOUT
Attach FRO 12 MHz to CLKOUT.

enumerator kPLL1_CLK0_to_CLKOUT
Attach PLL1_clk0 to CLKOUT.

enumerator kLPOSC_to_CLKOUT
Attach LP Oscillator to CLKOUT.

enumerator kUSB_PLL_to_CLKOUT
Attach USB_PLL to CLKOUT.

enumerator kNONE_to_CLKOUT
Attach NONE to CLKOUT.

enumerator kPLL0_to_ADC0
Attach PLL0 to ADC0.

enumerator kFRO_HF_to_ADC0
Attach FRO_HF to ADC0.

enumerator kFRO12M_to_ADC0
Attach FRO 12 MHz to ADC0.

enumerator kCLK_IN_to_ADC0
Attach Clk_in to ADC0.

enumerator kPLL1_CLK0_to_ADC0
Attach PLL1_clk0 to ADC0.

enumerator kUSB_PLL_to_ADC0
Attach USB PLL to ADC0.

enumerator kNONE_to_ADC0
Attach NONE to ADC0.

enumerator kPLL_DIV_to_FLEXCOMM0
Attach PLL_DIV to FLEXCOMM0.

enumerator kFRO12M_to_FLEXCOMM0
Attach FRO12M to FLEXCOMM0.

enumerator kFRO_HF_DIV_to_FLEXCOMM0
Attach FRO_HF_DIV to FLEXCOMM0.

enumerator kCLK_1M_to_FLEXCOMM0
Attach CLK_1MHz to FLEXCOMM0.

enumerator kUSB_PLL_to_FLEXCOMM0
Attach USB_PLL to FLEXCOMM0.

enumerator kLPOSC_to_FLEXCOMM0
Attach LP Oscillator to FLEXCOMM0.

enumerator kNONE_to_FLEXCOMM0
Attach NONE to FLEXCOMM0.

enumerator kPLL_DIV_to_FLEXCOMM1
Attach PLL_DIV to FLEXCOMM1.

enumerator kFRO12M_to_FLEXCOMM1
Attach FRO12M to FLEXCOMM1.

enumerator kFRO_HF_DIV_to_FLEXCOMM1
Attach FRO_HF_DIV to FLEXCOMM1.

enumerator kCLK_1M_to_FLEXCOMM1
Attach CLK_1MHz to FLEXCOMM1.

enumerator kUSB_PLL_to_FLEXCOMM1
Attach USB_PLL to FLEXCOMM1.

enumerator kLPOSC_to_FLEXCOMM1
Attach LP Oscillator to FLEXCOMM1.

enumerator kNONE_to_FLEXCOMM1
Attach NONE to FLEXCOMM1.

enumerator kPLL_DIV_to_FLEXCOMM2
Attach PLL_DIV to FLEXCOMM2.

enumerator kFRO12M_to_FLEXCOMM2
Attach FRO12M to FLEXCOMM2.

enumerator kFRO_HF_DIV_to_FLEXCOMM2
Attach FRO_HF_DIV to FLEXCOMM2.

enumerator kCLK_1M_to_FLEXCOMM2
Attach CLK_1MHz to FLEXCOMM2.

enumerator kUSB_PLL_to_FLEXCOMM2
Attach USB_PLL to FLEXCOMM2.

enumerator kLPOSC_to_FLEXCOMM2
Attach LP Oscillator to FLEXCOMM2.

enumerator kNONE_to_FLEXCOMM2
Attach NONE to FLEXCOMM2.

enumerator kPLL_DIV_to_FLEXCOMM3
Attach PLL_DIV to FLEXCOMM3.

enumerator kFRO12M_to_FLEXCOMM3
Attach FRO12M to FLEXCOMM3.

enumerator kFRO_HF_DIV_to_FLEXCOMM3
Attach FRO_HF_DIV to FLEXCOMM3.

enumerator kCLK_1M_to_FLEXCOMM3
Attach CLK_1MHz to FLEXCOMM3.

enumerator kUSB_PLL_to_FLEXCOMM3
Attach USB_PLL to FLEXCOMM3.

enumerator kLPOSC_to_FLEXCOMM3
Attach LP Oscillator to FLEXCOMM3.

enumerator kNONE_to_FLEXCOMM3
Attach NONE to FLEXCOMM3.

enumerator kPLL_DIV_to_FLEXCOMM4
Attach PLL_DIV to FLEXCOMM4.

enumerator kFRO12M_to_FLEXCOMM4
Attach FRO12M to FLEXCOMM4.

enumerator kFRO_HF_DIV_to_FLEXCOMM4
Attach FRO_HF_DIV to FLEXCOMM4.

enumerator kCLK_1M_to_FLEXCOMM4
Attach CLK_1MHz to FLEXCOMM4.

enumerator kUSB_PLL_to_FLEXCOMM4
Attach USB_PLL to FLEXCOMM4.

enumerator kLPOSC_to_FLEXCOMM4
Attach LP Oscillator to FLEXCOMM4.

enumerator kNONE_to_FLEXCOMM4
Attach NONE to FLEXCOMM4.

enumerator kPLL_DIV_to_FLEXCOMM5
Attach PLL_DIV to FLEXCOMM5.

enumerator kFRO12M_to_FLEXCOMM5
Attach FRO12M to FLEXCOMM5.

enumerator kFRO_HF_DIV_to_FLEXCOMM5
Attach FRO_HF_DIV to FLEXCOMM5.

enumerator kCLK_1M_to_FLEXCOMM5
Attach CLK_1MHz to FLEXCOMM5.

enumerator kUSB_PLL_to_FLEXCOMM5
Attach USB_PLL to FLEXCOMM5.

enumerator kLPOSC_to_FLEXCOMM5
Attach LP Oscillator to FLEXCOMM5.

enumerator kNONE_to_FLEXCOMM5
Attach NONE to FLEXCOMM5.

enumerator kPLL_DIV_to_FLEXCOMM6
Attach PLL_DIV to FLEXCOMM6.

enumerator kFRO12M_to_FLEXCOMM6
Attach FRO12M to FLEXCOMM6.

enumerator kFRO_HF_DIV_to_FLEXCOMM6
Attach FRO_HF_DIV to FLEXCOMM6.

enumerator kCLK_1M_to_FLEXCOMM6
Attach CLK_1MHz to FLEXCOMM6.

enumerator kUSB_PLL_to_FLEXCOMM6
Attach USB_PLL to FLEXCOMM6.

enumerator kLPOSC_to_FLEXCOMM6
Attach LP Oscillator to FLEXCOMM6.

enumerator kNONE_to_FLEXCOMM6
Attach NONE to FLEXCOMM6.

enumerator kPLL_DIV_to_FLEXCOMM7
Attach PLL_DIV to FLEXCOMM7.

enumerator kFRO12M_to_FLEXCOMM7
Attach FRO12M to FLEXCOMM7.

enumerator kFRO_HF_DIV_to_FLEXCOMM7
Attach FRO_HF_DIV to FLEXCOMM7.

enumerator kCLK_1M_to_FLEXCOMM7
Attach CLK_1MHz to FLEXCOMM7.

enumerator kUSB_PLL_to_FLEXCOMM7
Attach USB_PLL to FLEXCOMM7.

enumerator kLPOSC_to_FLEXCOMM7
Attach LP Oscillator to FLEXCOMM7.

enumerator kNONE_to_FLEXCOMM7
Attach NONE to FLEXCOMM7.

enumerator kPLL0_to_ADC1
Attach PLL0 to ADC1.

enumerator kFRO_HF_to_ADC1
Attach FRO_HF to ADC1.

enumerator kFRO12M_to_ADC1
Attach FRO12M to ADC1.

enumerator kCLK_IN_to_ADC1
Attach clk_in to ADC1.

enumerator kPLL1_CLK0_to_ADC1
Attach PLL1_clk0 to ADC1.

enumerator kUSB_PLL_to_ADC1
Attach USB PLL to ADC1.

enumerator kNONE_to_ADC1
Attach NONE to ADC1.

enumerator kPLL0_to_PLLCLKDIV
Attach PLL0 to PLLCLKDIV.

enumerator kPLL1_CLK0_to_PLLCLKDIV
Attach pll1_clk0 to PLLCLKDIV.

enumerator kNONE_to_PLLCLKDIV
Attach NONE to PLLCLKDIV.

enumerator kPLL0_to_I3C0FCLK
Attach PLL0 to I3C0FCLK.

enumerator kFRO_HF_to_I3C0FCLK
Attach FRO_HF to I3C0FCLK.

enumerator kCLK_1M_to_I3C0FCLK
Attach CLK_1M to I3C0FCLK.

enumerator kPLL1_CLK0_to_I3C0FCLK
Attach PLL1_clk0 to I3C0FCLK.

enumerator kUSB_PLL_to_I3C0FCLK
Attach USB PLL to I3C0FCLK.

enumerator kNONE_to_I3C0FCLK
Attach NONE to I3C0FCLK.

enumerator kPLL0_to_I3C0FCLKSTC
Attach PLL0 to I3C0FCLKSTC.

enumerator kFRO_HF_to_I3C0FCLKSTC
Attach FRO_HF to I3C0FCLKSTC.

enumerator kCLK_1M_to_I3C0FCLKSTC
Attach CLK_1M to I3C0FCLKSTC.

enumerator kPLL1_CLK0_to_I3C0FCLKSTC
Attach PLL1_clk0 to I3C0FCLKSTC.

enumerator kUSB_PLL_to_I3C0FCLKSTC
Attach USB PLL to I3C0FCLKSTC.

enumerator kNONE_to_I3C0FCLKSTC
Attach NONE to I3C0FCLKSTC.

enumerator kPLL0_to_I3C0FCLKSlow
Attach PLL0 to I3C0FCLKS.

enumerator kFRO_HF_to_I3C0FCLKSlow
Attach FRO_HF to I3C0FCLKS.

enumerator kCLK_1M_to_I3C0FCLKSlow
Attach CLK_1M to I3C0FCLKS.

enumerator kPLL1_CLK0_to_I3C0FCLKSlow
Attach PLL1_clk0 to I3C0FCLKS.

enumerator kUSB_PLL_to_I3C0FCLKSlow
Attach USB PLL to I3C0FCLKS.

enumerator kNONE_to_I3C0FCLKSlow
Attach NONE to I3C0FCLKS.

enumerator kFRO12M_to_MICFILF
Attach FRO_12M to MICFILE.

enumerator kPLL0_to_MICFILF
Attach PLL0 to MICFILE.

enumerator kCLK_IN_to_MICFILF
Attach Clk_in to MICFILE.

enumerator kFRO_HF_to_MICFILF
Attach FRO_HF to MICFILE.

enumerator kPLL1_CLK0_to_MICFILF
Attach PLL1_clk0 to MICFILE.

enumerator kSAI0_MCLK_IN_to_MICFILF
Attach SAI0_MCLK to MICFILE.

enumerator kUSB_PLL_to_MICFILF
Attach USB PLL to MICFILE.

enumerator kSAI1_MCLK_IN_to_MICFILF
Attach SAI1_MCLK to MICFILE.

enumerator kNONE_to_MICFILF
Attach NONE to MICFILE.

enumerator kPLL0_to_FLEXIO
Attach PLL0 to FLEXIO.

enumerator kCLK_IN_to_FLEXIO
Attach Clk_in to FLEXIO.

enumerator kFRO_HF_to_FLEXIO
Attach FRO_HF to FLEXIO.

enumerator kFRO12M_to_FLEXIO
Attach FRO_12M to FLEXIO.

enumerator kPLL1_CLK0_to_FLEXIO
Attach pll1_clk0 to FLEXIO.

enumerator kUSB_PLL_to_FLEXIO
Attach USB PLL to FLEXIO.

enumerator kNONE_to_FLEXIO
Attach NONE to FLEXIO.

enumerator kPLL0_to_FLEXCAN0
Attach PLL0 to FLEXCAN0.

enumerator kCLK_IN_to_FLEXCAN0
Attach Clk_in to FLEXCAN0.

enumerator kFRO_HF_to_FLEXCAN0
Attach FRO_HF to FLEXCAN0.

enumerator kPLL1_CLK0_to_FLEXCAN0
Attach pll1_clk0 to FLEXCAN0.

enumerator kUSB_PLL_to_FLEXCAN0
Attach USB PLL to FLEXCAN0.

enumerator kNONE_to_FLEXCAN0
Attach NONE to FLEXCAN0.

enumerator kPLL0_to_FLEXCAN1
Attach PLL0 to FLEXCAN1.

enumerator kCLK_IN_to_FLEXCAN1
Attach Clk_in to FLEXCAN1.

enumerator kFRO_HF_to_FLEXCAN1
Attach FRO_HF to FLEXCAN1.

enumerator kPLL1_CLK0_to_FLEXCAN1
Attach pll1_clk0 to FLEXCAN1.

enumerator kUSB_PLL_to_FLEXCAN1
Attach USB PLL to FLEXCAN1.

enumerator kNONE_to_FLEXCAN1
Attach NONE to FLEXCAN1.

enumerator kCLK_16K2_to_EWM0
Attach clk_16k[2] to EWM0.

enumerator kXTAL32K2_to_EWM0
Attach xtal32k[2] to EWM0.

enumerator kCLK_16K2_to_WDT1
Attach FRO16K clock 2 to WDT1.

enumerator kFRO_HF_DIV_to_WDT1
Attach FRO_HF_DIV to WDT1.

enumerator kCLK_1M_to_WDT1
Attach clk_1m to WDT1.

enumerator kCLK_1M_2_to_WDT1
Attach clk_1m to WDT1.

enumerator kCLK_16K2_to_OSTIMER
Attach clk_16k[2] to OSTIMER.

enumerator kXTAL32K2_to_OSTIMER
Attach xtal32k[2] to OSTIMER.

enumerator kCLK_1M_to_OSTIMER
Attach clk_1m to OSTIMER.

enumerator kNONE_to_OSTIMER
Attach NONE to OSTIMER.

enumerator kPLL0_to_CMP0F
Attach PLL0 to CMP0F.

enumerator kFRO_HF_to_CMP0F
Attach FRO_HF to CMP0F.

enumerator kFRO12M_to_CMP0F
Attach FRO_12M to CMP0F.

enumerator kCLK_IN_to_CMP0F
Attach Clk_in to CMP0F.

enumerator kPLL1_CLK0_to_CMP0F
Attach PLL1_clk0 to CMP0F.

enumerator kUSB_PLL_to_CMP0F
Attach USB PLL to CMP0F.

enumerator kNONE_to_CMP0F
Attach NONE to CMP0F.

enumerator kPLL0_to_CMP0RR
Attach PLL0 to CMP0RR.

enumerator kFRO_HF_to_CMP0RR
Attach FRO_HF to CMP0RR.

enumerator kFRO12M_to_CMP0RR
Attach FRO_12M to CMP0RR.

enumerator kCLK_IN_to_CMP0RR
Attach Clk_in to CMP0RR.

enumerator kPLL1_CLK0_to_CMP0RR
Attach PLL1_clk0 to CMP0RR.

enumerator kUSB_PLL_to_CMP0RR
Attach USB PLL to CMP0RR.

enumerator kNONE_to_CMP0RR
Attach NONE to CMP0RR.

enumerator kPLL0_to_CMP1F
Attach PLL0 to CMP1F.

enumerator kFRO_HF_to_CMP1F
Attach FRO_HF to CMP1F.

enumerator kFRO12M_to_CMP1F
Attach FRO_12M to CMP1F.

enumerator kCLK_IN_to_CMP1F
Attach Clk_in to CMP1F.

enumerator kPLL1_CLK0_to_CMP1F
Attach PLL1_clk0 to CMP1F.

enumerator kUSB_PLL_to_CMP1F
Attach USB PLL to CMP1F.

enumerator kNONE_to_CMP1F
Attach NONE to CMP1F.

enumerator kPLL0_to_CMP1RR
Attach PLL0 to CMP1RR.

enumerator kFRO_HF_to_CMP1RR
Attach FRO_HF to CMP1RR.

enumerator kFRO12M_to_CMP1RR
Attach FRO_12M to CMP1RR.

enumerator kCLK_IN_to_CMP1RR
Attach Clk_in to CMP1RR.

enumerator kPLL1_CLK0_to_CMP1RR
Attach PLL1_clk0 to CMP1RR.

enumerator kUSB_PLL_to_CMP1RR
Attach USB PLL to CMP1RR.

enumerator kNONE_to_CMP1RR
Attach NONE to CMP1RR.

enumerator kCLK_IN_to_UTICK
Attach Clk_in to UTICK.

enumerator kXTAL32K2_to_UTICK
Attach xtal32k[2] to UTICK.

enumerator kCLK_1M_to_UTICK
Attach clk_1m to UTICK.

enumerator kNONE_to_UTICK
Attach NONE to UTICK.

enumerator kPLL0_to_SAI0
Attach PLL0 to SAI0.

enumerator kCLK_IN_to_SAI0
Attach Clk_in to SAI0.

enumerator kFRO_HF_to_SAI0
Attach FRO_HF to SAI0.

enumerator kPLL1_CLK0_to_SAI0
Attach PLL1_clk0 to SAI0.

enumerator kUSB_PLL_to_SAI0
Attach USB PLL to SAI0.

enumerator kNONE_to_SAI0
Attach NONE to SAI0.

enumerator kPLL0_to_SAI1
Attach PLL0 to SAI1.

enumerator kCLK_IN_to_SAI1
Attach Clk_in to SAI1.

enumerator kFRO_HF_to_SAI1
Attach FRO_HF to SAI1.

enumerator kPLL1_CLK0_to_SAI1
Attach PLL1_clk0 to SAI1.

enumerator kUSB_PLL_to_SAI1
Attach USB PLL to SAI1.

enumerator kNONE_to_SAI1
Attach NONE to SAI1.

enumerator kPLL0_to_I3C1FCLK
Attach PLL0 to I3C1FCLK.

enumerator kFRO_HF_to_I3C1FCLK
Attach FRO_HF to I3C1FCLK.

enumerator kCLK_1M_to_I3C1FCLK
Attach CLK_1M to I3C1FCLK.

enumerator kPLL1_CLK0_to_I3C1FCLK
Attach PLL1_clk0 to I3C1FCLK.

enumerator kUSB_PLL_to_I3C1FCLK
Attach USB PLL to I3C1FCLK.

enumerator kNONE_to_I3C1FCLK
Attach NONE to I3C1FCLK.

enumerator kPLL0_to_I3C1FCLKSTC
Attach PLL0 to I3C1FCLKSTC.

enumerator kFRO_HF_to_I3C1FCLKSTC
Attach FRO_HF to I3C1FCLKSTC.

enumerator kCLK_1M_to_I3C1FCLKSTC
Attach CLK_1M to I3C1FCLKSTC.

enumerator kPLL1_CLK0_to_I3C1FCLKSTC
Attach PLL1_clk0 to I3C1FCLKSTC.

enumerator kUSB_PLL_to_I3C1FCLKSTC
Attach USB PLL to I3C1FCLKSTC.

enumerator kNONE_to_I3C1FCLKSTC
Attach NONE to I3C1FCLKSTC.

enumerator kPLL0_to_I3C1FCLKSlow
Attach PLL0 to I3C1FCLKS.

enumerator kFRO_HF_to_I3C1FCLKSlow
Attach FRO_HF to I3C1FCLKS.

enumerator kCLK_1M_to_I3C1FCLKSlow
Attach CLK_1M to I3C1FCLKS.

enumerator kPLL1_CLK0_to_I3C1FCLKSlow
Attach PLL1_clk0 to I3C1FCLKS.

enumerator kUSB_PLL_to_I3C1FCLKSlow
Attach USB PLL to I3C1FCLKS.

enumerator kNONE_to_I3C1FCLKSlow
Attach NONE to I3C1FCLKS.

enumerator kNONE_to_NONE
Attach NONE to NONE.

enum _clock_div_name

Clock dividers.

Values:

enumerator kCLOCK_DivSystickClk0

Systick Clk0 Divider.

enumerator kCLOCK_DivTraceClk

Trace Clk Divider.

enumerator kCLOCK_DivSlowClk

SLOW CLK Divider.

enumerator kCLOCK_DivAhbClk

Ahb Clk Divider.

enumerator kCLOCK_DivClkOut

ClkOut Clk Divider.

enumerator kCLOCK_DivFrohfClk

Frohf Clk Divider.

enumerator kCLOCK_DivWdt0Clk

Wdt0 Clk Divider.

enumerator kCLOCK_DivAdc0Clk

Adc0 Clk Divider.

enumerator kCLOCK_DivPllClk

Pll Clk Divider.

enumerator kCLOCK_DivCtimer0Clk

Ctimer0 Clk Divider.

enumerator kCLOCK_DivCtimer1Clk

Ctimer1 Clk Divider.

enumerator kCLOCK_DivCtimer2Clk

Ctimer2 Clk Divider.

enumerator kCLOCK_DivCtimer3Clk

Ctimer3 Clk Divider.

enumerator kCLOCK_DivCtimer4Clk

Ctimer4 Clk Divider.

enumerator kCLOCK_DivPLL1Clk0

PLL1 Clk0 Divider.

enumerator kCLOCK_DivPLL1Clk1

Pll1 Clk1 Divider.

enumerator kCLOCK_DivUtickClk

Utick Clk Divider.

enumerator kCLOCK_DivFrg

CLKOUT FRG Clk Divider.

enumerator kCLOCK_DivAdc1Clk

Adc1 Clk Divider.

enumerator kCLOCK_DivI3c0FClk
I3C0 FClk Divider.

enumerator kCLOCK_DivMicfilFClk
MICFILCLK Divider.

enumerator kCLOCK_DivFlexioClk
Flexio Clk Divider.

enumerator kCLOCK_DivFlexcan0Clk
Flexcan0 Clk Divider.

enumerator kCLOCK_DivFlexcan1Clk
Flexcan1 Clk Divider.

enumerator kCLOCK_DivWdt1Clk
Wdt1 Clk Divider.

enumerator kCLOCK_DivCmp0FClk
Cmp0 FClk Divider.

enumerator kCLOCK_DivCmp0rrClk
Cmp0rr Clk Divider.

enumerator kCLOCK_DivCmp1FClk
Cmp1 FClk Divider.

enumerator kCLOCK_DivCmp1rrClk
Cmp1rr Clk Divider.

enumerator kCLOCK_DivFlexcom0Clk
Flexcom0 Clk Divider.

enumerator kCLOCK_DivFlexcom1Clk
Flexcom1 Clk Divider.

enumerator kCLOCK_DivFlexcom2Clk
Flexcom2 Clk Divider.

enumerator kCLOCK_DivFlexcom3Clk
Flexcom3 Clk Divider.

enumerator kCLOCK_DivFlexcom4Clk
Flexcom4 Clk Divider.

enumerator kCLOCK_DivFlexcom5Clk
Flexcom5 Clk Divider.

enumerator kCLOCK_DivFlexcom6Clk
Flexcom6 Clk Divider.

enumerator kCLOCK_DivFlexcom7Clk
Flexcom7 Clk Divider.

enumerator kCLOCK_DivSai0Clk
Sai0 Clk Divider.

enumerator kCLOCK_DivSai1Clk
Sai1 Clk Divider.

enumerator kCLOCK_DivI3c1FClk
I3C1 FClk Divider.

enum _osc32k_clk_gate_id

OSC32K clock gate.

Values:

enumerator kCLOCK_Osc32kToVbat

OSC32K[0] to VBAT domain.

enumerator kCLOCK_Osc32kToVsys

OSC32K[1] to VSYS domain.

enumerator kCLOCK_Osc32kToWake

OSC32K[2] to WAKE domain.

enumerator kCLOCK_Osc32kToMain

OSC32K[3] to MAIN domain.

enumerator kCLOCK_Osc32kToAll

OSC32K to VBAT,VSYS,WAKE,MAIN domain.

enum _clk16k_clk_gate_id

CLK16K clock gate.

Values:

enumerator kCLOCK_Clk16KToVbat

Clk16k[0] to VBAT domain.

enumerator kCLOCK_Clk16KToVsys

Clk16k[1] to VSYS domain.

enumerator kCLOCK_Clk16KToWake

Clk16k[2] to WAKE domain.

enumerator kCLOCK_Clk16KToMain

Clk16k[3] to MAIN domain.

enumerator kCLOCK_Clk16KToAll

Clk16k to VBAT,VSYS,WAKE,MAIN domain.

enum _clock_ctrl_enable

system clocks enable controls

Values:

enumerator kCLOCK_FRO1MHZ_CLK_ENA

Enables FRO_1MHz clock for clock muxing in clock gen.

enumerator kCLOCK_CLKIN_ENA

Enables clk_in clock for MICD, EMVSIM0/1, CAN0/1, I3C0/1, SAI0/1, clkout

enumerator kCLOCK_FRO_HF_ENA

Enables FRO HF clock for the Frequency Measure module.

enumerator kCLOCK_FRO12MHZ_ENA

Enables the FRO_12MHz clock for the Flash, LPTIMER0/1, and Frequency Measurement modules.

enumerator kCLOCK_FRO1MHZ_ENA

Enables the FRO_1MHz clock for RTC module and for UTICK.

enumerator kCLOCK_CLKIN_ENA_FM_USBH_LPT

Enables the clk_in clock for the Frequency Measurement, USB HS and LPTIMER0/1 modules.

enum _clock_usb_phy_src

Source of the USB HS PHY.

Values:

enumerator kCLOCK_Usbphy480M

Use 480M.

enum _scg_status

SCG status return codes.

Values:

enumerator kStatus_SCG_Busy

Clock is busy.

enumerator kStatus_SCG_InvalidSrc

Invalid source.

enum _firc_trim_mode

firc trim mode.

Values:

enumerator kSCG_FircTrimNonUpdate

Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure trim_config_t.

enumerator kSCG_FircTrimUpdate

Trim enable and trim value update enable. In this mode, the trim value is auto update.

enum _firc_trim_src

firc trim source.

Values:

enumerator kSCG_FircTrimSrcUsb0

USB0 start of frame (1kHz).

enumerator kSCG_FircTrimSrcSysOsc

System OSC.

enumerator kSCG_FircTrimSrcRtcOsc

RTC OSC (32.768 kHz).

enum _sirc_trim_mode

sirc trim mode.

Values:

enumerator kSCG_SircTrimNonUpdate

Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure trim_config_t.

enumerator kSCG_SircTrimUpdate

Trim enable and trim value update enable. In this mode, the trim value is auto update.

enum _sirc_trim_src

sirc trim source.

Values:

enumerator kSCG_SircTrimSrcSysOsc
System OSC.

enumerator kSCG_SircTrimSrcRtcOsc
RTC OSC (32.768 kHz).

enum _scg_sosc_monitor_mode
SCG system OSC monitor mode.

Values:

enumerator kSCG_SysOscMonitorDisable
Monitor disabled.

enumerator kSCG_SysOscMonitorInt
Interrupt when the SOSC error is detected.

enumerator kSCG_SysOscMonitorReset
Reset when the SOSC error is detected.

enum _scg_rosc_monitor_mode
SCG ROSC monitor mode.

Values:

enumerator kSCG_RoscMonitorDisable
Monitor disabled.

enumerator kSCG_RoscMonitorInt
Interrupt when the RTC OSC error is detected.

enumerator kSCG_RoscMonitorReset
Reset when the RTC OSC error is detected.

enum _scg_upll_monitor_mode
SCG UPLL monitor mode.

Values:

enumerator kSCG_UpllMonitorDisable
Monitor disabled.

enumerator kSCG_UpllMonitorInt
Interrupt when the UPLL error is detected.

enumerator kSCG_UpllMonitorReset
Reset when the UPLL error is detected.

enum _scg_pll0_monitor_mode
SCG PLL0 monitor mode.

Values:

enumerator kSCG_Pll0MonitorDisable
Monitor disabled.

enumerator kSCG_Pll0MonitorInt
Interrupt when the PLL0 Clock error is detected.

enumerator kSCG_Pll0MonitorReset
Reset when the PLL0 Clock error is detected.

enum _scg_pll1_monitor_mode

SCG PLL1 monitor mode.

Values:

enumerator kSCG_Pll1MonitorDisable

Monitor disabled.

enumerator kSCG_Pll1MonitorInt

Interrupt when the PLL1 Clock error is detected.

enumerator kSCG_Pll1MonitorReset

Reset when the PLL1 Clock error is detected.

enum _vbat_osc_xtal_cap

The enumerator of internal capacitance of OSC's XTAL pin.

Values:

enumerator kVBAT_OscXtal0pFCap

The internal capacitance for XTAL pin is 0pF.

enumerator kVBAT_OscXtal2pFCap

The internal capacitance for XTAL pin is 2pF.

enumerator kVBAT_OscXtal4pFCap

The internal capacitance for XTAL pin is 4pF.

enumerator kVBAT_OscXtal6pFCap

The internal capacitance for XTAL pin is 6pF.

enumerator kVBAT_OscXtal8pFCap

The internal capacitance for XTAL pin is 8pF.

enumerator kVBAT_OscXtal10pFCap

The internal capacitance for XTAL pin is 10pF.

enumerator kVBAT_OscXtal12pFCap

The internal capacitance for XTAL pin is 12pF.

enumerator kVBAT_OscXtal14pFCap

The internal capacitance for XTAL pin is 14pF.

enumerator kVBAT_OscXtal16pFCap

The internal capacitance for XTAL pin is 16pF.

enumerator kVBAT_OscXtal18pFCap

The internal capacitance for XTAL pin is 18pF.

enumerator kVBAT_OscXtal20pFCap

The internal capacitance for XTAL pin is 20pF.

enumerator kVBAT_OscXtal22pFCap

The internal capacitance for XTAL pin is 22pF.

enumerator kVBAT_OscXtal24pFCap

The internal capacitance for XTAL pin is 24pF.

enumerator kVBAT_OscXtal26pFCap

The internal capacitance for XTAL pin is 26pF.

enumerator kVBAT_OscXtal28pFCap

The internal capacitance for XTAL pin is 28pF.

enumerator kVBAT_OscXtal30pFCap

The internal capacitance for XTAL pin is 30pF.

enum _vbat_osc_extal_cap

The enumerator of internal capacitance of OSC's EXTAL pin.

Values:

enumerator kVBAT_OscExtal0pFCap

The internal capacitance for EXTAL pin is 0pF.

enumerator kVBAT_OscExtal2pFCap

The internal capacitance for EXTAL pin is 2pF.

enumerator kVBAT_OscExtal4pFCap

The internal capacitance for EXTAL pin is 4pF.

enumerator kVBAT_OscExtal6pFCap

The internal capacitance for EXTAL pin is 6pF.

enumerator kVBAT_OscExtal8pFCap

The internal capacitance for EXTAL pin is 8pF.

enumerator kVBAT_OscExtal10pFCap

The internal capacitance for EXTAL pin is 10pF.

enumerator kVBAT_OscExtal12pFCap

The internal capacitance for EXTAL pin is 12pF.

enumerator kVBAT_OscExtal14pFCap

The internal capacitance for EXTAL pin is 14pF.

enumerator kVBAT_OscExtal16pFCap

The internal capacitance for EXTAL pin is 16pF.

enumerator kVBAT_OscExtal18pFCap

The internal capacitance for EXTAL pin is 18pF.

enumerator kVBAT_OscExtal20pFCap

The internal capacitance for EXTAL pin is 20pF.

enumerator kVBAT_OscExtal22pFCap

The internal capacitance for EXTAL pin is 22pF.

enumerator kVBAT_OscExtal24pFCap

The internal capacitance for EXTAL pin is 24pF.

enumerator kVBAT_OscExtal26pFCap

The internal capacitance for EXTAL pin is 26pF.

enumerator kVBAT_OscExtal28pFCap

The internal capacitance for EXTAL pin is 28pF.

enumerator kVBAT_OscExtal30pFCap

The internal capacitance for EXTAL pin is 30pF.

enum _vbat_osc_fine_adjustment_value

The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by modifying the automatic gain control (AGC).

Values:

enumerator kVBAT_OscCoarseAdjustment05

enumerator kVBAT_OscCoarseAdjustment10

enumerator kVBAT_OscCoarseAdjustment18

enumerator kVBAT_OscCoarseAdjustment33

enum _run_mode

The active run mode (voltage level).

Values:

enumerator kMD_Mode

Midvoltage (1.0 V).

enumerator kSD_Mode

Normal voltage (1.1 V).

enumerator kOD_Mode

Overdrive voltage (1.2 V).

enum _vbat_osc_init_trim

The enumerator of Initialization Trim.

Values:

enumerator kVBAT_OscInitTrim8000ms

Configures the start-up time of the oscillator to 8s.

enumerator kVBAT_OscInitTrim4000ms

Configures the start-up time of the oscillator to 4s.

enumerator kVBAT_OscInitTrim2000ms

Configures the start-up time of the oscillator to 2s.

enumerator kVBAT_OscInitTrim1000ms

Configures the start-up time of the oscillator to 1s.

enumerator kVBAT_OscInitTrim500ms

Configures the start-up time of the oscillator to 0.5s.

enumerator kVBAT_OscInitTrim250ms

Configures the start-up time of the oscillator to 0.25s.

enumerator kVBAT_OscInitTrim125ms

Configures the start-up time of the oscillator to 0.125s.

enumerator kVBAT_OscInitTrimHalfms

Configures the start-up time of the oscillator to 0.5ms.

enum _vbat_osc_cap_trim

The enumerator of Capacitor Trim.

Values:

enumerator kVBAT_OscCapTrimDefault

enumerator kVBAT_OscCapTrim1us

enumerator kVBAT_OscCapTrim2us

enumerator kVBAT_OscCapTrim2andhalfus

enum _vbat_osc_dly_trim

The enumerator of Delay Trim.

Values:

enumerator kVBAT_OscDlyTrim0

P current 9(nA) and N Current 6(nA).

enumerator kVBAT_OscDlyTrim1

P current 13(nA) and N Current 6(nA).

enumerator kVBAT_OscDlyTrim3

P current 4(nA) and N Current 6(nA).

enumerator kVBAT_OscDlyTrim4

P current 9(nA) and N Current 4(nA).

enumerator kVBAT_OscDlyTrim5

P current 13(nA) and N Current 4(nA).

enumerator kVBAT_OscDlyTrim6

P current 4(nA) and N Current 4(nA).

enumerator kVBAT_OscDlyTrim7

P current 9(nA) and N Current 2(nA).

enumerator kVBAT_OscDlyTrim8

P current 13(nA) and N Current 2(nA).

enumerator kVBAT_OscDlyTrim9

P current 4(nA) and N Current 2(nA).

enum _vbat_osc_cap2_trim

The enumerator of CAP2_TRIM.

Values:

enumerator kVBAT_OscCap2Trim0

enumerator kVBAT_OscCap2Trim1

enum _vbat_osc_cmp_trim

The enumerator of Comparator Trim.

Values:

enumerator kVBAT_OscCmpTrim760mv

enumerator kVBAT_OscCmpTrim770mv

enumerator kVBAT_OscCmpTrim740mv

enum _vbat_osc_mode_en

The enumerator of configures Crystal Oscillator mode..

Values:

enumerator kVBAT_OscNormalModeEnable

enumerator kVBAT_OscStartupModeEnable

enumerator kVBAT_OscLowpowerModeEnable

enum _pll_clk_src

PLL clock source.

Values:

enumerator kPll_ClkSrcSysOsc

System OSC.

enumerator kPll_ClkSrcFirc
Fast IRC.

enumerator kPll_ClkSrcRosc
RTC OSC.

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 $\approx$ 3.9 - 7.8 kHz)

enumerator kSS_MF_384
Nss $\approx$ 384 (fm $\approx$ 5.2 - 10.4 kHz)

enumerator kSS_MF_256
Nss = 256 (fm $\approx$ 7.8 - 15.6 kHz)

enumerator kSS_MF_128
Nss = 128 (fm $\approx$ 15.6 - 31.3 kHz)

enumerator kSS_MF_64
Nss = 64 (fm $\approx$ 32.3 - 64.5 kHz)

enumerator kSS_MF_32
Nss = 32 (fm $\approx$ 62.5 - 125 kHz)

enumerator kSS_MF_24
Nss $\approx$ 24 (fm $\approx$ 83.3 - 166.6 kHz)

enumerator kSS_MF_16
Nss = 16 (fm $\approx$ 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 $\approx$ 1

enumerator kSS_MR_K1_5
k $\approx$ 1.5

enumerator kSS_MR_K2
k $\approx$ 2

enumerator kSS_MR_K3
k $\approx$ 3

enumerator kSS_MR_K4
k $\approx$ 4

enumerator kSS_MR_K6
k $\approx$ 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_OutputError

PLL output rate error

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

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 _osc32k_clk_gate_id osc32k_clk_gate_id_t
    OSC32K clock gate.

typedef enum _clk16k_clk_gate_id clk16k_clk_gate_id_t
    CLK16K clock gate.

typedef enum _clock_ctrl_enable clock_ctrl_enable_t
    system clocks enable controls

typedef enum _clock_usb_phy_src clock_usb_phy_src_t
    Source of the USB HS PHY.

typedef enum _firc_trim_mode firc_trim_mode_t
    firc trim mode.

typedef enum _firc_trim_src firc_trim_src_t
    firc trim source.

typedef struct _firc_trim_config firc_trim_config_t
    firc trim configuration.

typedef enum _sirc_trim_mode sirc_trim_mode_t
    sirc trim mode.

typedef enum _sirc_trim_src sirc_trim_src_t
    sirc trim source.

typedef struct _sirc_trim_config sirc_trim_config_t
    sirc trim configuration.

typedef enum _scg_sosc_monitor_mode scg_sosc_monitor_mode_t
    SCG system OSC monitor mode.

typedef enum _scg_rosc_monitor_mode scg_rosc_monitor_mode_t
    SCG ROSC monitor mode.

typedef enum _scg_upll_monitor_mode scg_upll_monitor_mode_t
    SCG UPLL monitor mode.

typedef enum _scg_pll0_monitor_mode scg_pll0_monitor_mode_t
    SCG PLL0 monitor mode.

typedef enum _scg_pll1_monitor_mode scg_pll1_monitor_mode_t
    SCG PLL1 monitor mode.

typedef enum _vbat_osc_xtal_cap vbat_osc_xtal_cap_t
    The enumerator of internal capacitance of OSC's XTAL pin.

typedef enum _vbat_osc_extal_cap vbat_osc_extal_cap_t
    The enumerator of internal capacitance of OSC's EXTAL pin.

typedef enum _vbat_osc_fine_adjustment_value vbat_osc_coarse_adjustment_value_t
    The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by
    modifying the automatic gain control (AGC).

typedef struct _vbat_osc_config vbat_osc_config_t
    The structure of oscillator configuration.

typedef enum _run_mode run_mode_t
    The active run mode (voltage level).
```

typedef enum *_vbat_osc_init_trim* vbat_osc_init_trim_t

The enumerator of Initialization Trim.

typedef enum *_vbat_osc_cap_trim* vbat_osc_cap_trim_t

The enumerator of Capacitor Trim.

typedef enum *_vbat_osc_dly_trim* vbat_osc_dly_trim_t

The enumerator of Delay Trim.

typedef enum *_vbat_osc_cap2_trim* vbat_osc_cap2_trim_t

The enumerator of CAP2_TRIM.

typedef enum *_vbat_osc_cmp_trim* vbat_osc_cmp_trim_t

The enumerator of Comparator Trim.

typedef enum *_vbat_osc_mode_en* vbat_osc_mode_en_t

The enumerator of configures Crystal Oscillator mode..

typedef struct *_osc_32k_config* osc_32k_config_t

The structure of oscillator configuration.

typedef enum *_pll_clk_src* pll_clk_src_t

PLL clock source.

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.

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_SetupFROHFClocking(uint32_t iFreq)

Initialize the Core clock to given frequency (48 or 144 MHz). This function turns on FIRC and select the given frequency as the source of fro_hf.

Parameters

- *iFreq* – : Desired frequency (must be one of CLK_FRO_44MHZ or CLK_FRO_144MHZ)

Returns

returns success or fail status.

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_SetupExtRefClocking(uint32_t iFreq)

Initialize the external reference 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_SetupOsc32KClocking(uint32_t id)

Initialize the XTAL32/EXTAL32 input clock to given frequency.

Parameters

- *id* – : OSC 32 kHz output clock to specified modules, it should use osc32k_clk_gate_id_t value

Returns

returns success or fail status.

void CLOCK_GetDefaultOsc32KConfig(*osc_32k_config_t* *config)

Get default XTAL32/EXTAL32 clock configuration structure. This function initializes the osc 32k configuration structure to a default value. The default values are: config->initTrim = kVBAT_OscInitTrim500ms; config->capTrim = kVBAT_OscCapTrimDefault; config->dlyTrim = kVBAT_OscDlyTrim5; config->cap2Trim = kVBAT_OscCap2Trim0; config->cmpTrim = kVBAT_OscCmpTrim760mv; config->mode = kVBAT_OscNormalModeEnable; config->xtalCap = kVBAT_OscXtal24pFCap; config->extalCap = kVBAT_OscExtal22pFCap; config->ampGain = kVBAT_OscCoarseAdjustment05; config->id = kCLOCK_Osc32kToVbat;.

Parameters

- *config* – Pointer to a configuration structure

status_t CLOCK_SetupOsc32KClockingConfig(*osc_32k_config_t* config)

Initialize the OSC 32K with user-defined settings.

Parameters

- config – : OSC 32K configuration structure

Returns

returns success or fail status.

status_t CLOCK_SetupClk16KClocking(*uint32_t* id)

Initialize the FRO16K input clock to given frequency.

Parameters

- id – : FRO 16 kHz output clock to specified modules, it should use *clk16k_clk_gate_id_t* value

Returns

returns success or fail status.

status_t CLOCK_FROHFTrimConfig(*firc_trim_config_t* config)

Setup FROHF trim.

Parameters

- config – : FROHF trim value

Returns

returns success or fail status.

status_t CLOCK_FRO12MTrimConfig(*sirc_trim_config_t* config)

Setup FRO 12M trim.

Parameters

- config – : FRO 12M trim value

Returns

returns success or fail status.

void CLOCK_SetSysOscMonitorMode(*scg_sosc_monitor_mode_t* mode)

Sets the system OSC monitor mode.

This function sets the system OSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

void CLOCK_SetRoscMonitorMode(*scg_rosc_monitor_mode_t* mode)

Sets the ROSC monitor mode.

This function sets the ROSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

void CLOCK_SetUpllMonitorMode(*scg_upll_monitor_mode_t* mode)

Sets the UPLL monitor mode.

This function sets the UPLL monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

`void CLOCK_SetPll0MonitorMode(scg_pll0_monitor_mode_t mode)`

Sets the PLL0 monitor mode.

This function sets the PLL0 monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

`void CLOCK_SetPll1MonitorMode(scg_pll1_monitor_mode_t mode)`

Sets the PLL1 monitor mode.

This function sets the PLL1 monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

`void VBAT_SetOscConfig(VBAT_Type *base, const vbat_osc_config_t *config)`

Config 32k Crystal Oscillator.

Parameters

- base – VBAT peripheral base address.
- config – The pointer to the structure *vbat_osc_config_t*.

`status_t CLOCK_SetFLASHAccessCyclesForFreq(uint32_t system_freq_hz, run_mode_t mode)`

Set the additional number of wait-states added to account for the ratio of system clock period to flash access time during full speed power mode.

Parameters

- system_freq_hz – : Input frequency
- mode – : Active run mode (voltage level).

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)`

Setup peripheral clock dividers.

Parameters

- div_name – : Clock divider name

- `divided_by_value` – Value to be divided

Returns

Nothing

`uint32_t` `CLOCK_GetClkDiv(clock_div_name_t div_name)`

Get peripheral clock dividers.

Parameters

- `div_name` – : Clock divider name

Returns

peripheral clock dividers

`void` `CLOCK_HaltClkDiv(clock_div_name_t div_name)`

Halt peripheral clock dividers.

Parameters

- `div_name` – : Clock divider name

Returns

Nothing

`void` `CLOCK_SetupClockCtrl(uint32_t mask)`

system clocks enable controls.

Parameters

- `mask` – : system clocks enable value, it should use `clock_ctrl_enable_t` value

Returns

Nothing

`uint32_t` `CLOCK_GetFreq(clock_name_t clockName)`

Return Frequency of selected clock.

Returns

Frequency of selected clock

`uint32_t` `CLOCK_GetCoreSysClkFreq(void)`

Return Frequency of core.

Returns

Frequency of the core

`uint32_t` `CLOCK_GetCTimerClkFreq(uint32_t id)`

Return Frequency of CTimer functional Clock.

Returns

Frequency of CTimer functional Clock

`uint32_t` `CLOCK_GetAdcClkFreq(uint32_t id)`

Return Frequency of Adc Clock.

Returns

Frequency of Adc.

`uint32_t` `CLOCK_GetLPFlexCommClkFreq(uint32_t id)`

Return Frequency of LPFlexComm Clock.

Returns

Frequency of LPFlexComm Clock

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 PLLCLKDIV.

Returns

Frequency of PLLCLKDIV Clock

uint32_t CLOCK_GetI3cClkFreq(uint32_t id)

Return Frequency of I3C function Clock.

Returns

Frequency of I3C function Clock

uint32_t CLOCK_GetMicfilClkFreq(void)

Return Frequency of MICFIL Clock.

Returns

Frequency of MICFIL.

uint32_t CLOCK_GetFlexioClkFreq(void)

Return Frequency of FLEXIO.

Returns

Frequency of FLEXIO Clock

uint32_t CLOCK_GetFlexcanClkFreq(uint32_t id)

Return Frequency of FLEXCAN.

Returns

Frequency of FLEXCAN Clock

uint32_t CLOCK_GetEwm0ClkFreq(void)

Return Frequency of EWM0 Clock.

Returns

Frequency of EWM0.

uint32_t CLOCK_GetWdtClkFreq(uint32_t id)

Return Frequency of Watchdog.

Returns

Frequency of Watchdog

uint32_t CLOCK_GetOstimerClkFreq(void)

Return Frequency of OSTIMER.

Returns

Frequency of OSTIMER Clock

uint32_t CLOCK_GetCmpFClkFreq(uint32_t id)

Return Frequency of CMP Function Clock.

Returns

Frequency of CMP Function.

uint32_t CLOCK_GetCmpRRClkFreq(uint32_t id)

Return Frequency of CMP Round Robin Clock.

Returns

Frequency of CMP Round Robin.

uint32_t CLOCK_GetUtickClkFreq(void)

Return Frequency of UTICK Clock.

Returns

Frequency of UTICK Clock.

uint32_t CLOCK_GetSaiClkFreq(uint32_t id)

Return Frequency of SAI Clock.

Returns

Frequency of SAI Clock.

void CLOCK_SetupSaiMclk(uint32_t id, uint32_t iFreq)

Initialize the SAI MCLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns

Nothing

void CLOCK_SetupSaiTxBclk(uint32_t id, uint32_t iFreq)

Initialize the SAI TX BCLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns

Nothing

void CLOCK_SetupSaiRxBclk(uint32_t id, uint32_t iFreq)

Initialize the SAI RX BCLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns

Nothing

uint32_t CLOCK_GetSaiMclkFreq(uint32_t id)

Return Frequency of SAI MCLK.

Returns

Frequency of SAI MCLK

uint32_t CLOCK_GetSaiTxBclkFreq(uint32_t id)

Return Frequency of SAI TX BCLK.

Returns

Frequency of SAI TX BCLK

uint32_t CLOCK_GetSaiRxBclkFreq(uint32_t id)

Return Frequency of SAI RX BCLK.

Returns

Frequency of SAI RX BCLK

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_GetExtUpllFreq(void)

Gets the external UPLL frequency.

Returns

The frequency of the external UPLL.

void CLOCK_SetExtUpllFreq(uint32_t freq)

Sets the external UPLL frequency.

Parameters

- The – frequency of external UPLL.

__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

uint32_t CLOCK_GetPLLOutFromSetup(*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

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

Set PLL 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_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_EnableOstimer32kClock(void)

Enable the OSTIMER 32k clock.

Returns

Nothing

bool CLOCK_EnableUsbhsPhyPllClock(*clock_usb_phy_src_t* src, uint32_t freq)

brief Enable USB HS PHY PLL clock.

This function enables the internal 480MHz USB PHY PLL clock.

param src USB HS PHY PLL clock source. param freq The frequency specified by src. retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

void CLOCK_DisableUsbhsPhyPllClock(void)

brief Disable USB HS PHY PLL clock.

This function disables USB HS PHY PLL clock.

bool CLOCK_EnableUsbhsClock(void)

brief Enable USB HS clock. retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

status_t CLOCK_FIRCAutoTrimWithSOF(void)

FIRC Auto Trim With SOF.

Returns

returns success or fail status.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.0.0.

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.

FMC_CLOCKS

Clock ip name array for FMC.

INPUTMUX_CLOCKS

Clock ip name array for INPUTMUX.

GPIO_CLOCKS

Clock ip name array for GPIO.

GDET_CLOCKS

Clock ip name array for GDET.

PINT_CLOCKS

Clock ip name array for PINT.

DMA_CLOCKS

Clock ip name array for DMA.

EDMA_CLOCKS

Clock gate name array for EDMA.

CRC_CLOCKS

Clock ip name array for CRC.

WWDT_CLOCKS

Clock ip name array for WWDT.

LPADC_CLOCKS

Clock ip name array for LPADC.

MRT_CLOCKS

Clock ip name array for MRT.

OSTIMER_CLOCKS

Clock ip name array for OSTIMER.

UTICK_CLOCKS

Clock ip name array for UTICK.

LP_FLEXCOMM_CLOCKS

Clock ip name array for LP_FLEXCOMM.

LPUART_CLOCKS

Clock ip name array for LPUART.

LPI2C_CLOCKS

Clock ip name array for LPI2C.

LPSPI_CLOCKS

Clock ip name array for LSPI.

CTIMER_CLOCKS

Clock ip name array for CTIMER.

FREQME_CLOCKS

Clock ip name array for FREQME.

PUF_CLOCKS

Clock ip name array for PUF.

VREF_CLOCKS

Clock ip name array for VREF.

PWM_CLOCKS

Clock ip name array for PWM.

QDC_CLOCKS

Clock ip name array for QDC.

FLEXIO_CLOCKS

Clock ip name array for FLEXIO.

FLEXCAN_CLOCKS

Clock ip name array for FLEXCAN.

I3C_CLOCKS

Clock ip name array for I3C.

USDHC_CLOCKS

Clock ip name array for USDHC.

SAI_CLOCKS

Clock ip name array for SAI.

RTC_CLOCKS

Clock ip name array for RTC.

PDM_CLOCKS

Clock ip name array for PDM.

ERM_CLOCKS

Clock ip name array for ERM.

EIM_CLOCKS

Clock ip name array for EIM.

TRNG_CLOCKS

Clock ip name array for TRNG.

LPCMP_CLOCKS

 Clock ip name array for LPCMP.

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_CLKOUTCLKSEL
CM_ADC0CLKSEL
CM_FCCLKSEL0
CM_FCCLKSEL1
CM_FCCLKSEL2
CM_FCCLKSEL3
CM_FCCLKSEL4
CM_FCCLKSEL5
CM_FCCLKSEL6
CM_FCCLKSEL7
CM_ADC1CLKSEL
CM_PLLCLKDIVSEL
CM_I3C0FCLKSEL
CM_MICFILFCLKSEL
CM_FLEXIOCLKSEL
CM_FLEXCAN0CLKSEL
CM_FLEXCAN1CLKSEL
CM_EWM0CLKSEL
CM_WDT1CLKSEL
CM_OSTIMERCLKSEL
CM_CMP0FCLKSEL
CM_CMP0RRCLKSEL
CM_CMP1FCLKSEL
CM_CMP1RRCLKSEL
CM_UTICKCLKSEL
CM_SAI0CLKSEL
CM_SAI1CLKSEL
CM_I3C1FCLKSEL
CM_SCGRCCRSCCLKSEL

PLL_CONFIGFLAG_FORCENOFRACT

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

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

firc_trim_mode_t trimMode

Trim mode.

firc_trim_src_t trimSrc

Trim source.

uint16_t trimDiv

Divider of SOSC.

uint8_t trimCoar

Trim coarse value; Irrelevant if trimMode is kSCG_TrimUpdate.

uint8_t trimFine

Trim fine value; Irrelevant if trimMode is kSCG_TrimUpdate.

sirc_trim_mode_t trimMode

Trim mode.

sirc_trim_src_t trimSrc

Trim source.

uint16_t trimDiv

Divider of SOSC.

uint8_t cltrim

Trim coarse value; Irrelevant if trimMode is kSCG_TrimUpdate.

uint8_t ccotrim

Trim fine value; Irrelevant if trimMode is kSCG_TrimUpdate.

bool enableInternalCapBank

enable/disable the internal capacitance bank.

bool enableCrystalOscillatorBypass

enable/disable the crystal oscillator bypass.

vbat_osc_xtal_cap_t xtalCap

The internal capacitance for the OSC XTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.

vbat_osc_extal_cap_t extalCap

The internal capacitance for the OSC EXTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.

vbat_osc_coarse_adjustment_value_t coarseAdjustment

32kHz crystal oscillator amplifier coarse adjustment value.

vbat_osc_init_trim_t initTrim

vbat_osc_cap_trim_t capTrim

vbat_osc_dly_trim_t dlyTrim

vbat_osc_cap2_trim_t cap2Trim

vbat_osc_cmp_trim_t cmpTrim

vbat_osc_mode_en_t mode

vbat_osc_xtal_cap_t xtalCap

vbat_osc_extal_cap_t extalCap

vbat_osc_coarse_adjustment_value_t ampGain

osc32k_clk_gate_id_t id

uint32_t desiredRate

Desired PLL rate in Hz

uint32_t inputSource

PLL input source

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 APLLCTRL

uint32_t pllndiv

PLL N Divider register APLLNDIV

uint32_t pllpdiv

PLL P Divider register APLLPDIV

uint32_t pllmdiv

PLL M Divider register APLLMDIV

uint32_t pllsscg[2]

PLL Spread Spectrum Control registers APLLSSCG

uint32_t pllRate

Actual PLL rate

struct _firc_trim_config

#include <fsl_clock.h> firc trim configuration.

struct __sirc_trim_config

#include <fsl_clock.h> sirc trim configuration.

struct __vbat_osc_config

#include <fsl_clock.h> The structure of oscillator configuration.

struct __osc_32k_config

#include <fsl_clock.h> The structure of oscillator configuration.

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.

2.4 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()

2.5 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

2.6 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA_Init(EDMA_Type *base, const edma_config_t *config)

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure. All emda enabled request will be cleared in this function.

Note: This function enables the minor loop map feature.

Parameters

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see “edma_config_t”.

void EDMA_Deinit(*EDMA_Type* *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

void EDMA_InstallTCD(*EDMA_Type* *base, uint32_t channel, *edma_tcd_t* *tcd)

Push content of TCD structure into hardware TCD register.

Parameters

- base – EDMA peripheral base address.
- channel – EDMA channel number.
- tcd – Point to TCD structure.

void EDMA_GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableContinuousLinkMode = false;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
```

Parameters

- config – A pointer to the eDMA configuration structure.

void EDMA_InitChannel(*EDMA_Type* *base, uint32_t channel, *edma_channel_config_t* *channelConfig)

EDMA Channel initialization.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelConfig – pointer to user's eDMA4 channel config structure, see *edma_channel_config_t* for detail.

static inline void EDMA_SetChannelMemoryAttribute(*EDMA_Type* *base, uint32_t channel, *edma_channel_memory_attribute_t* writeAttribute, *edma_channel_memory_attribute_t* readAttribute)

Set channel memory attribute.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- writeAttribute – Attributes associated with a write transaction.
- readAttribute – Attributes associated with a read transaction.

static inline void EDMA_SetChannelSignExtension(*EDMA_Type* *base, uint32_t channel, uint8_t position)

Set channel sign extension.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- position – A non-zero value specifying the sign extend bit position. If 0, sign extension is disabled.

```
static inline void EDMA_SetChannelSwapSize(EDMA_Type *base, uint32_t channel,  
                                           edma_channel_swap_size_t swapSize)
```

Set channel swap size.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- swapSize – Swap occurs with respect to the specified transfer size. If 0, swap is disabled.

```
static inline void EDMA_SetChannelAccessType(EDMA_Type *base, uint32_t channel,  
                                             edma_channel_access_type_t  
                                             channelAccessType)
```

Set channel access type.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelAccessType – eDMA4's transactions type on the system bus when the channel is active.

```
static inline void EDMA_SetChannelMux(EDMA_Type *base, uint32_t channel, uint32_t  
                                     channelRequestSource)
```

Set channel request source.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- channelRequestSource – eDMA hardware service request source for the channel. User need to use the `dma_request_source_t` type as the input parameter. Note that devices may use other enum type to express dma request source and User can fined it in SOC header or `fsl_edma_soc.h`.

```
static inline uint32_t EDMA_GetChannelSystemBusInformation(EDMA_Type *base, uint32_t  
                                                         channel)
```

Gets the channel identification and attribute information on the system bus interface.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of the channel system bus information. Users need to use the `_edma_channel_sys_bus_info` type to decode the return variables.

```
static inline void EDMA_EnableChannelMasterIDReplication(EDMA_Type *base, uint32_t  
                                                         channel, bool enable)
```

Set channel master ID replication.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – true is enable, false is disable.

```
static inline void EDMA_SetChannelProtectionLevel(EDMA_Type *base, uint32_t channel,  
                                                edma_channel_protection_level_t level)
```

Set channel security level.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- level – security level.

```
void EDMA_ResetChannel(EDMA_Type *base, uint32_t channel)
```

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

Note: This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
void EDMA_SetTransferConfig(EDMA_Type *base, uint32_t channel, const  
                           edma_transfer_config_t *config, edma_tcd_t *nextTcd)
```

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_t config;  
edma_tcd_t tcd;  
config.srcAddr = ..;  
config.destAddr = ..;  
...  
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

Note: If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_ResetChannel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.

- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_SetMinorOffsetConfig(EDMA_Type *base, uint32_t channel, const  
                               edma_minor_offset_config_t *config)
```

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_SetChannelPreemptionConfig(EDMA_Type *base, uint32_t channel, const  
                                     edma_channel_preemption_config_t *config)
```

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

```
void EDMA_SetChannelLink(EDMA_Type *base, uint32_t channel, edma_channel_link_type_t  
                        type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
 - kEDMA_LinkNone
 - kEDMA_MinorLink
 - kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
void EDMA_SetBandWidth(EDMA_Type *base, uint32_t channel, edma_bandwidth_t  
                      bandWidth)
```

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
 - kEDMABandwidthStallNone
 - kEDMABandwidthStall4Cycle
 - kEDMABandwidthStall8Cycle

```
void EDMA_SetModulo(EDMA_Type *base, uint32_t channel, edma_modulo_t srcModulo,  
                  edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA_EnableAsyncRequest(EDMA_Type *base, uint32_t channel, bool enable)
```

Enables an async request for the eDMA transfer.

Parameters

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

```
static inline void EDMA_EnableAutoStopRequest(EDMA_Type *base, uint32_t channel, bool  
                                             enable)
```

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

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

```
void EDMA_EnableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)
```

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA_DisableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)`

Disables the interrupt source for the eDMA transfer.

Parameters

- `base` – eDMA peripheral base address.
- `channel` – eDMA channel number.
- `mask` – The mask of the interrupt source to be set. Use the defined `edma_interrupt_enable_t` type.

`void EDMA_SetMajorOffsetConfig(EDMA_Type *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)`

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- `base` – eDMA peripheral base address.
- `channel` – edma channel number.
- `sourceOffset` – source address offset will be applied to source address after major loop done.
- `destOffset` – destination address offset will be applied to source address after major loop done.

`void EDMA_ConfigChannelSoftwareTCD(edma_tcd_t *tcd, const edma_transfer_config_t *transfer)`

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA_ConfigChannelSoftwareTCDExt`

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- `tcd` – Pointer to the TCD structure.
- `transfer` – channel transfer configuration pointer.

`void EDMA_TcdReset(edma_tcd_t *tcd)`

Sets all fields to default values for the TCD structure.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA_TcdResetExt`

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- `tcd` – Pointer to the TCD structure.

```
void EDMA__TcdSetTransferConfig(edma_tcd_t *tcd, const edma_transfer_config_t *config,  
                               edma_tcd_t *nextTcd)
```

Configures the eDMA TCD transfer attribute.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetTransferConfigExt

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {  
    ...  
}  
edma_tcd_t tcd __aligned(32);  
edma_tcd_t nextTcd __aligned(32);  
EDMA__TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA__TcdSetMinorOffsetConfig(edma_tcd_t *tcd, const edma_minor_offset_config_t  
                                  *config)
```

Configures the eDMA TCD minor offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetMinorOffsetConfigExt

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

```
void EDMA__TcdSetChannelLink(edma_tcd_t *tcd, edma_channel_link_type_t type, uint32_t  
                             linkedChannel)
```

Sets the channel link for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetChannelLinkExt

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- `tcd` – Point to the TCD structure.
- `type` – Channel link type, it can be one of:
 - `kEDMA_LinkNone`
 - `kEDMA_MinorLink`
 - `kEDMA_MajorLink`
- `linkedChannel` – The linked channel number.

static inline void EDMA_TcdSetBandWidth(*edma_tcd_t* *tcd, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetBandWidthExt

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
 - `kEDMABandwidthStallNone`
 - `kEDMABandwidthStall4Cycle`
 - `kEDMABandwidthStall8Cycle`

void EDMA_TcdSetModulo(*edma_tcd_t* *tcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetModuloExt

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- `tcd` – A pointer to the TCD structure.
- `srcModulo` – A source modulo value.
- `destModulo` – A destination modulo value.

static inline void EDMA_TcdEnableAutoStopRequest(*edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdEnableAutoStopRequestExt

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- `tcd` – A pointer to the TCD structure.
- `enable` – The command to enable (true) or disable (false).

void EDMA__TcdEnableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA__TcdEnableInterruptsExt

Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

void EDMA__TcdDisableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Disables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA__TcdDisableInterruptsExt

Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

void EDMA__TcdSetMajorOffsetConfig(*edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA__TcdSetMajorOffsetConfigExt

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- `tcd` – A point to the TCD structure.
- `sourceOffset` – source address offset will be applied to source address after major loop done.
- `destOffset` – destination address offset will be applied to source address after major loop done.

void EDMA__ConfigChannelSoftwareTCDExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, const *edma_transfer_config_t* *transfer)

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.

- transfer – channel transfer configuration pointer.

void EDMA_TcdResetExt(*EDMA_Type* *base, *edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.

void EDMA_TcdSetTransferConfigExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_TcdSetMinorOffsetConfigExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, const *edma_minor_offset_config_t* *config)

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.

- `config` – A pointer to the minor offset configuration structure.

```
void EDMA__TcdSetChannelLinkExt(EDMA_Type *base, edma_tcd_t *tcd,  
                               edma_channel_link_type_t type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Point to the TCD structure.
- `type` – Channel link type, it can be one of:
 - `kEDMA_LinkNone`
 - `kEDMA_MinorLink`
 - `kEDMA_MajorLink`
- `linkedChannel` – The linked channel number.

```
static inline void EDMA__TcdSetBandWidthExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                             edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
 - `kEDMABandwidthStallNone`
 - `kEDMABandwidthStall4Cycle`
 - `kEDMABandwidthStall8Cycle`

```
void EDMA__TcdSetModuloExt(EDMA_Type *base, edma_tcd_t *tcd, edma_modulo_t srcModulo,  
                           edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – A pointer to the TCD structure.
- `srcModulo` – A source modulo value.

- destModulo – A destination modulo value.

```
static inline void EDMA__TcdEnableAutoStopRequestExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                                    bool enable)
```

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

```
void EDMA__TcdEnableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)
```

Enables the interrupt source for the eDMA TCD.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA__TcdDisableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)
```

Disables the interrupt source for the eDMA TCD.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA__TcdSetMajorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, int32_t  
                                     sourceOffset, int32_t destOffset)
```

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

```
static inline void EDMA__EnableChannelRequest(EDMA_Type *base, uint32_t channel)
```

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
static inline void EDMA_DisableChannelRequest(EDMA_Type *base, uint32_t channel)
```

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
static inline void EDMA_TriggerChannelStart(EDMA_Type *base, uint32_t channel)
```

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

```
uint32_t EDMA_GetRemainingMajorLoopCount(EDMA_Type *base, uint32_t channel)
```

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

- a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTES(initially configured)
-

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

Major loop count which has not been transferred yet for the current TCD.

```
static inline uint32_t EDMA_GetErrorStatusFlags(EDMA_Type *base)
```

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

The mask of error status flags. Users need to use the `_edma_error_status_flags` type to decode the return variables.

```
uint32_t EDMA_GetChannelStatusFlags(EDMA_Type *base, uint32_t channel)
```

Gets the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

`void EDMA_ClearChannelStatusFlags(EDMA_Type *base, uint32_t channel, uint32_t mask)`

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined `_edma_channel_status_flags` type.

`status_t EDMA_CreateHandle(edma_handle_t *handle, EDMA_Type *base, uint32_t channel)`

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

Return values

- `kStatus_Success` –
- `kStatus_InvalidArgument` –

`void EDMA_InstallTCDMemory(edma_handle_t *handle, edma_tcd_t *tcdPool, uint32_t tcdSize)`

Installs the TCDs memory pool into the eDMA handle.

This function is called after the `EDMA_CreateHandle` to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface `EDMA_SubmitTransfer`.

Parameters

- handle – eDMA handle pointer.
- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

`void EDMA_SetCallback(edma_handle_t *handle, edma_callback callback, void *userData)`

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.

- `userData` – A parameter for the callback function.

```
void EDMA__PrepareTransferConfig(edma_transfer_config_t *config, void *srcAddr, uint32_t  
                                srcWidth, int16_t srcOffset, void *destAddr, uint32_t  
                                destWidth, int16_t destOffset, uint32_t bytesEachRequest,  
                                uint32_t transferBytes)
```

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE). User can check if 128 bytes support is available for specific instance by `FSL_FEATURE_EDMA_INSTANCE_SUPPORT_128_BYTES_TRANSFERn`.

Parameters

- `config` – The user configuration structure of type `edma_transfer_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – source address offset.
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.

```
void EDMA__PrepareTransfer(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth,  
                           void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest,  
                           uint32_t transferBytes, edma_transfer_type_t type)
```

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- `config` – The user configuration structure of type `edma_transfer_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `type` – eDMA transfer type.

```
void EDMA__PrepareTransferTCD(edma_handle_t *handle, edma_tcd_t *tcd, void *srcAddr,
                             uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t
                             destWidth, int16_t destOffset, uint32_t bytesEachRequest,
                             uint32_t transferBytes, edma_tcd_t *nextTcd)
```

Prepares the eDMA transfer content descriptor.

This function prepares the transfer content descriptor structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- handle – eDMA handle pointer.
- tcd – Pointer to eDMA transfer content descriptor structure.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- nextTcd – eDMA transfer linked TCD address.

```
status_t EDMA__SubmitTransferTCD(edma_handle_t *handle, edma_tcd_t *tcd)
```

Submits the eDMA transfer content descriptor.

This function submits the eDMA transfer request according to the transfer content descriptor. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA__InstallTCDMemory before.

Typical user case:

a. submit single transfer

```
edma_tcd_t tcd;
EDMA__PrepareTransferTCD(handle, tcd, ...)
EDMA__SubmitTransferTCD(handle, tcd)
EDMA__StartTransfer(handle)
```

b. submit static link transfer,

```
edma_tcd_t tcd[2];
EDMA__PrepareTransferTCD(handle, &tcd[0], ...)
EDMA__PrepareTransferTCD(handle, &tcd[1], ...)
EDMA__SubmitTransferTCD(handle, &tcd[0])
EDMA__StartTransfer(handle)
```

c. submit dynamic link transfer

```
edma_tcd_t tcdpool[2];
EDMA__InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
edma_tcd_t tcd;
```

(continues on next page)

(continued from previous page)

```
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

d. submit loop transfer

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ..., &tcd[1])
EDMA_PrepareTransferTCD(handle, &tcd[1], ..., &tcd[0])
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

Parameters

- handle – eDMA handle pointer.
- tcd – Pointer to eDMA transfer content descriptor structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

status_t EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDEMemory before.

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

status_t EDMA_SubmitLoopTransfer(*edma_handle_t* *handle, *edma_transfer_config_t* *transfer, *uint32_t* transferLoopCount)

Submits the eDMA scatter gather transfer configurations.

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
edma_channel_transfer_config_t transfer[2];
EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

Note: Application should check the return value of this function to avoid transfer request submit failed

Parameters

- handle – eDMA handle pointer
- transfer – pointer to user's eDMA channel configure structure, see `edma_channel_transfer_config_t` for detail
- transferLoopCount – the count of the transfer ring, if loop count is 1, that means that the one will link to itself.

Return values

- `kStatus_Success` – It means submit transfer request succeed
- `kStatus_EDMA_Busy` – channel is in busy status
- `kStatus_InvalidArgument` – Invalid Argument

`void EDMA_StartTransfer(edma_handle_t *handle)`

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

`void EDMA_StopTransfer(edma_handle_t *handle)`

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call `EDMA_StartTransfer()` again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

`void EDMA_AbortTransfer(edma_handle_t *handle)`

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

`static inline uint32_t EDMA_GetUnusedTCDNumber(edma_handle_t *handle)`

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

```
static inline uint32_t EDMA_GetNextTCDAddress(edma_handle_t *handle)
```

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- *handle* – DMA handle pointer.

Returns

The next TCD address.

```
void EDMA_HandleIRQ(edma_handle_t *handle)
```

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As *sga* and *sga_index* are calculated based on the *DLAST_SGA* bitfield lies in the *TCD_CSR* register, the *sga_index* in this case should be 2 (*DLAST_SGA* of TCD[1] stores the address of TCD[2]). Thus, the “*tcdUsed*” updated should be (*tcdUsed* - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and *tcdUsed* updated are identical for them. *tcdUsed* are both 0 in this case as no TCD to be loaded.

See the “eDMA basic data flow” in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- *handle* – eDMA handle pointer.

```
FSL_EDMA_DRIVER_VERSION
```

eDMA driver version

Version 2.10.7.

```
_edma_transfer_status eDMA transfer status
```

Values:

```
enumerator kStatus_EDMA_QueueFull
```

TCD queue is full.

```
enumerator kStatus_EDMA_Busy
```

Channel is busy and can't handle the transfer request.

```
enum _edma_transfer_size
```

eDMA transfer configuration

Values:

```
enumerator kEDMA_TransferSize1Bytes
```

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes

Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA_TransferSize128Bytes

Source/Destination data transfer size is 128 bytes every time

enum _edma_modulo

eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable

Disable modulo

enumerator kEDMA_Modulo2bytes

Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes

Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes

Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes

Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes

Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes

Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes

Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes

Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes

Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes

Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes

Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone

No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle

eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle

eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type

Channel link type.

Values:

enumerator kEDMA_LinkNone

No channel link

enumerator kEDMA_MinorLink

Channel link after each minor loop

enumerator kEDMA_MajorLink

Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.

Values:

enumerator kEDMA_DoneFlag

DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag

eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag

eDMA interrupt flag, set while an interrupt occurred of this channel

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag

Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag

Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag

Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag

NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag

Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag

Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag

Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag

Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag

Error channel number of the cancelled channel number

enumerator kEDMA_TransferCanceledFlag

Transfer cancelled

enumerator kEDMA_ValidFlag

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

_edma_interrupt_enable eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable

Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable

Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable

Enable interrupt while major count to half value.

enum _edma_transfer_type

eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory

Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory

Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral

Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral

Transfer from Peripheral to peripheral

enum edma_channel_memory_attribute

eDMA channel memory attribute

Values:

enumerator kEDMA_ChannelNoWriteNoReadNoCacheNoBuffer

No write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadNoCacheBufferable

No write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableNoBuffer

No write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableBufferable

No write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheNoBuffer

No write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheBufferable

No write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableNoBuffer

No write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableBufferable

No write allocate, read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheNoBuffer

write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheBufferable

write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableNoBuffer

write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableBufferable

write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheNoBuffer

write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheBufferable

write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadCacheableNoBuffer

write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadCacheableBufferable

write allocate, read allocate, cacheable, bufferable.

enum _edma_channel_swap_size

eDMA4 channel swap size

Values:

enumerator kEDMA_ChannelSwapDisabled

Swap is disabled.

enumerator kEDMA_ChannelReadWith8bitSwap

Swap occurs with respect to the read 8bit.

enumerator kEDMA_ChannelReadWith16bitSwap

Swap occurs with respect to the read 16bit.

enumerator kEDMA_ChannelReadWith32bitSwap

Swap occurs with respect to the read 32bit.

enumerator kEDMA_ChannelWriteWith8bitSwap

Swap occurs with respect to the write 8bit.

enumerator kEDMA_ChannelWriteWith16bitSwap

Swap occurs with respect to the write 16bit.

enumerator kEDMA_ChannelWriteWith32bitSwap

Swap occurs with respect to the write 32bit.

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

enumerator kEDMA_PrivilegedAccessLevel

Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA_MasterId

DMA's master ID when channel is active and master ID replication is enabled.

enum _edma_channel_access_type

eDMA4 channel access type

Values:

enumerator kEDMA_ChannelDataAccess

Data access for eDMA4 transfers.

enumerator kEDMA_ChannelInstructionAccess

Instruction access for eDMA4 transfers.

enum _edma_channel_protection_level

eDMA4 channel protection level

Values:

enumerator kEDMA_ChannelProtectionLevelUser

user protection level for eDMA transfers.

enumerator kEDMA_ChannelProtectionLevelPrivileged

Privileged protection level eDMA transfers.

typedef enum _edma_transfer_size edma_transfer_size_t

eDMA transfer configuration

typedef enum _edma_modulo edma_modulo_t

eDMA modulo configuration

typedef enum _edma_bandwidth edma_bandwidth_t

Bandwidth control.

typedef enum _edma_channel_link_type edma_channel_link_type_t

Channel link type.

typedef enum _edma_transfer_type edma_transfer_type_t

eDMA transfer type

typedef struct _edma_channel_Preemption_config edma_channel_Preemption_config_t

eDMA channel priority configuration

typedef struct _edma_minor_offset_config edma_minor_offset_config_t

eDMA minor offset configuration

typedef enum edma_channel_memory_attribute edma_channel_memory_attribute_t

eDMA channel memory attribute

typedef enum _edma_channel_swap_size edma_channel_swap_size_t

eDMA4 channel swap size

typedef enum _edma_channel_access_type edma_channel_access_type_t

eDMA4 channel access type

typedef enum _edma_channel_protection_level edma_channel_protection_level_t

eDMA4 channel protection level

typedef struct _edma_channel_config edma_channel_config_t

eDMA4 channel configuration

```
typedef edma_core_tcd_t edma_tcd_t  
    eDMA TCD.
```

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

```
typedef struct _edma_transfer_config edma_transfer_config_t  
    edma4 channel transfer configuration
```

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

Note: User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
 - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
 - c. the totalBytes should align with the bytesEachRequest
 - d. the srcAddr should align with the srcWidthOfEachTransfer
 - e. the dstAddr should align with the dstWidthOfEachTransfer
 - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
 - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
-

```
typedef struct _edma_config edma_config_t  
    eDMA global configuration structure.
```

```
typedef void (*edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone,  
    uint32_t tcDs)
```

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Param handle

EDMA handle pointer, users shall not touch the values inside.

Param userData

The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

Param transferDone

If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

Param tcDs

How many tcDs are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcDs are finished between the last callback and this.

```
typedef struct _edma_handle edma_handle_t
```

EDMA transfer handle structure

```
FSL_EDMA_DRIVER_EDMA4
```

EDMA driver name

```
EDMA_ALLOCATE_TCD(name, number)
```

Macro used for allocate edma TCD.

```
DMA_DCHPRI_INDEX(channel)
```

Compute the offset unit from DCHPRI3.

```
struct _edma_channel_Preemption_config
```

#include <fsl_edma.h> EDMA channel priority configuration

Public Members

```
bool enableChannelPreemption
```

If true: a channel can be suspended by other channel with higher priority

```
bool enablePreemptAbility
```

If true: a channel can suspend other channel with low priority

```
uint8_t channelPriority
```

Channel priority

```
struct _edma_minor_offset_config
```

#include <fsl_edma.h> EDMA minor offset configuration

Public Members

```
bool enableSrcMinorOffset
```

Enable(true) or Disable(false) source minor loop offset.

```
bool enableDestMinorOffset
```

Enable(true) or Disable(false) destination minor loop offset.

```
uint32_t minorOffset
```

Offset for a minor loop mapping.

```
struct _edma_channel_config
```

#include <fsl_edma.h> EDMA4 channel configuration

Public Members

edma_channel_Preemption_config_t channelPreemptionConfig
channel preemption configuration

edma_channel_memory_attribute_t channelReadMemoryAttribute
channel memory read attribute configuration

edma_channel_memory_attribute_t channelWriteMemoryAttribute
channel memory write attribute configuration

edma_channel_swap_size_t channelSwapSize
channel swap size configuration

edma_channel_access_type_t channelAccessType
channel access type configuration

uint8_t channelDataSignExtensionBitPosition
channel data sign extension bit position configuration

uint32_t channelRequestSource
hardware service request source for the channel

bool enableMasterIDReplication
enable master ID replication

edma_channel_protection_level_t protectionLevel
protection level

struct __edma_transfer_config

#include <fsl_edma.h> edma4 channel transfer configuration

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enableChannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

Note: User should pay attention to the transfer size alignment limitation

- the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
- the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
- the totalBytes should align with the bytesEachRequest
- the srcAddr should align with the srcWidthOfEachTransfer

- e. the dstAddr should align with the dstWidthOfEachTransfer
 - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
 - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
-

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

Sign-extended offset value in byte unit applied to the current source address to form the next-state value as each source read is completed

int16_t destOffset

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

uint32_t minorLoopBytes

bytes in each minor loop or each request range: 1 - (2^{30} - 1) when minor loop mapping is enabled range: 1 - (2^{10} - 1) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - (2^{32} - 1) when minor loop mapping is disabled

uint32_t majorLoopCounts

minor loop counts in each major loop, should be 1 at least for each transfer range: (0 - (2^{15} - 1)) when minor loop channel link is disabled range: (0 - (2^9 - 1)) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajorLoop * bytesEachMinorLoop

uint16_t enabledInterruptMask

channel interrupt to enable, can be OR'ed value of _edma_interrupt_enable

edma_modulo_t srcAddrModulo

source circular data queue range

int32_t srcMajorLoopOffset

source major loop offset

edma_modulo_t dstAddrModulo

destination circular data queue range

int32_t dstMajorLoopOffset

destination major loop offset

bool enableSrcMinorLoopOffset

enable source minor loop offset

bool enableDstMinorLoopOffset

enable dest minor loop offset

`int32_t` minorLoopOffset
burst offset, the offset will be applied after minor loop update

`bool` enableChannelMajorLoopLink
channel link when major loop complete

`uint32_t` majorLoopLinkChannel
major loop link channel number

`bool` enableChannelMinorLoopLink
channel link when minor loop complete

`uint32_t` minorLoopLinkChannel
minor loop link channel number

`edma_tcd_t` *linkTCD
pointer to the link transfer control descriptor

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

Public Members

`bool` enableMasterIdReplication
Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

`bool` enableGlobalChannelLink
Enable(true) channel linking is available and controlled by each channel's link settings.

`bool` enableHaltOnError
Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

`bool` enableDebugMode
Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

`bool` enableRoundRobinArbitration
Enable(true) channel linking is available and controlled by each channel's link settings.

`edma_channel_config_t` *channelConfig[1]
channel preemption configuration

`struct` _edma_handle
#include <fsl_edma.h> eDMA transfer handle structure

Public Members

`edma_callback` callback
Callback function for major count exhausted.

`void` *userData
Callback function parameter.

`EDMA_ChannelType` *channelBase
eDMA peripheral channel base address.

`EDMA_Type` *base
eDMA peripheral base address

EDMA_TCDType *tcdBase

eDMA peripheral tcd base address.

edma_tcd_t *tcdPool

Pointer to memory stored TCDs.

uint32_t channel

eDMA channel number.

volatile int8_t header

The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail

The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed

The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize

The total number of TCD slots in the queue.

2.7 eDMA core Driver

enum _edma_tcd_type

eDMA tcd flag type

Values:

enumerator kEDMA_EDMA4Flag

Data access for eDMA4 transfers.

enumerator kEDMA_EDMA5Flag

Instruction access for eDMA4 transfers.

typedef struct _edma_core_mp edma_core_mp_t

edma core channel struture definition

typedef struct _edma_core_channel edma_core_channel_t

edma core channel struture definition

typedef enum _edma_tcd_type edma_tcd_type_t

eDMA tcd flag type

typedef struct _edma5_core_tcd edma5_core_tcd_t

edma5 core TCD struture definition

typedef struct _edma4_core_tcd edma4_core_tcd_t

edma4 core TCD struture definition

typedef struct _edma_core_tcd edma_core_tcd_t

edma core TCD struture definition

typedef edma_core_channel_t EDMA_ChannelType

EDMA typedef.

typedef edma_core_tcd_t EDMA_TCDType

typedef void EDMA_Type

DMA_CORE_MP_CSR_EDBG_MASK
DMA_CORE_MP_CSR_ERCA_MASK
DMA_CORE_MP_CSR_HAE_MASK
DMA_CORE_MP_CSR_HALT_MASK
DMA_CORE_MP_CSR_GCLC_MASK
DMA_CORE_MP_CSR_GMRC_MASK
DMA_CORE_MP_CSR_EDBG(x)
DMA_CORE_MP_CSR_ERCA(x)
DMA_CORE_MP_CSR_HAE(x)
DMA_CORE_MP_CSR_HALT(x)
DMA_CORE_MP_CSR_GCLC(x)
DMA_CORE_MP_CSR_GMRC(x)
DMA_CSR_INTMAJOR_MASK
DMA_CSR_INTHALF_MASK
DMA_CSR_DREQ_MASK
DMA_CSR_ESG_MASK
DMA_CSR_BWC_MASK
DMA_CSR_BWC(x)
DMA_CSR_START_MASK
DMA_CITER_ELINKNO_CITER_MASK
DMA_BITER_ELINKNO_BITER_MASK
DMA_CITER_ELINKNO_CITER_SHIFT
DMA_CITER_ELINKYES_CITER_MASK
DMA_CITER_ELINKYES_CITER_SHIFT
DMA_ATTR_SMOD_MASK
DMA_ATTR_DMOD_MASK
DMA_CITER_ELINKNO_ELINK_MASK
DMA_CSR_MAJORELINK_MASK
DMA_BITER_ELINKYES_ELINK_MASK
DMA_CITER_ELINKYES_ELINK_MASK
DMA_CSR_MAJORLINKCH_MASK
DMA_BITER_ELINKYES_LINKCH_MASK
DMA_CITER_ELINKYES_LINKCH_MASK

DMA_NBYTES_MLOFFYES_MLOFF_MASK
DMA_NBYTES_MLOFFYES_DMLOE_MASK
DMA_NBYTES_MLOFFYES_SMLOE_MASK
DMA_NBYTES_MLOFFNO_NBYTES_MASK
DMA_ATTR_DMOD(x)
DMA_ATTR_SMOD(x)
DMA_BITER_ELINKYES_LINKCH(x)
DMA_CITER_ELINKYES_LINKCH(x)
DMA_NBYTES_MLOFFYES_MLOFF(x)
DMA_NBYTES_MLOFFYES_DMLOE(x)
DMA_NBYTES_MLOFFYES_SMLOE(x)
DMA_NBYTES_MLOFFNO_NBYTES(x)
DMA_NBYTES_MLOFFYES_NBYTES(x)
DMA_ATTR_DSIZE(x)
DMA_ATTR_SSIZE(x)
DMA_CSR_DREQ(x)
DMA_CSR_MAJORLINKCH(x)
DMA_CH_MATTR_WCACHE(x)
DMA_CH_MATTR_RCACHE(x)
DMA_CH_CSR_SIGNEXT_MASK
DMA_CH_CSR_SIGNEXT_SHIFT
DMA_CH_CSR_SWAP_MASK
DMA_CH_CSR_SWAP_SHIFT
DMA_CH_SBR_INSTR_MASK
DMA_CH_SBR_INSTR_SHIFT
DMA_CH_SBR_EMI_MASK
DMA_CH_SBR_EMI_SHIFT
DMA_CH_MUX_SOURCE(x)
DMA_ERR_DBE_FLAG
 DMA error flag.
DMA_ERR_SBE_FLAG
DMA_ERR_SGE_FLAG
DMA_ERR_NCE_FLAG

DMA_ERR_DOE_FLAG
DMA_ERR_DAE_FLAG
DMA_ERR_SOE_FLAG
DMA_ERR_SAE_FLAG
DMA_ERR_ERRCHAN_FLAG
DMA_ERR_ECX_FLAG
DMA_ERR_FLAG
DMA_CLEAR_DONE_STATUS(base, channel)
 get/clear DONE bit
DMA_GET_DONE_STATUS(base, channel)
DMA_ENABLE_ERROR_INT(base, channel)
 enable/disable error interrupt
DMA_DISABLE_ERROR_INT(base, channel)
DMA_CLEAR_ERROR_STATUS(base, channel)
 get/clear error status
DMA_GET_ERROR_STATUS(base, channel)
DMA_CLEAR_INT_STATUS(base, channel)
 get/clear INT status
DMA_GET_INT_STATUS(base, channel)
DMA_ENABLE_MAJOR_INT(base, channel)
 enable/disable MAJOR/HALF INT
DMA_ENABLE_HALF_INT(base, channel)
DMA_DISABLE_MAJOR_INT(base, channel)
DMA_DISABLE_HALF_INT(base, channel)
EDMA_TCD_ALIGN_SIZE
 EDMA tcd align size.
EDMA_CORE_BASE(base)
 EDMA base address convert macro.
EDMA_MP_BASE(base)
EDMA_CHANNEL_BASE(base, channel)
EDMA_TCD_BASE(base, channel)
EDMA_TCD_TYPE(x)
 EDMA TCD type macro.
EDMA_TCD_SADDR(tcd, flag)
 EDMA TCD address convert macro.
EDMA_TCD_SOFF(tcd, flag)
EDMA_TCD_ATTR(tcd, flag)

EDMA_TCD_NBYTES(tcd, flag)

EDMA_TCD_SLAST(tcd, flag)

EDMA_TCD_DADDR(tcd, flag)

EDMA_TCD_DOFF(tcd, flag)

EDMA_TCD_CITER(tcd, flag)

EDMA_TCD_DLAST_SGA(tcd, flag)

EDMA_TCD_CSR(tcd, flag)

EDMA_TCD_BITER(tcd, flag)

struct __edma_core_mp

#include <fsl_edma_core.h> edma core channel struture definition

Public Members

__IO uint32_t MP_CSR

Channel Control and Status, array offset: 0x10000, array step: 0x10000

__IO uint32_t MP_ES

Channel Error Status, array offset: 0x10004, array step: 0x10000

struct __edma_core_channel

#include <fsl_edma_core.h> edma core channel struture definition

Public Members

__IO uint32_t CH_CSR

Channel Control and Status, array offset: 0x10000, array step: 0x10000

__IO uint32_t CH_ES

Channel Error Status, array offset: 0x10004, array step: 0x10000

__IO uint32_t CH_INT

Channel Interrupt Status, array offset: 0x10008, array step: 0x10000

__IO uint32_t CH_SBR

Channel System Bus, array offset: 0x1000C, array step: 0x10000

__IO uint32_t CH_PRI

Channel Priority, array offset: 0x10010, array step: 0x10000

struct __edma5_core_tcd

#include <fsl_edma_core.h> edma5 core TCD struture definition

Public Members

__IO uint32_t SADDR

SADDR register, used to save source address

__IO uint32_t SADDR_HIGH

SADDR HIGH register, used to save source address


```

__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t SLAST_SDA_HIGH
    SLAST SDA HIGH register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint32_t DADDR_HIGH
    DADDR HIGH register, used for destination address
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint32_t DLAST_SGA_HIGH
    DLASTSGA HIGH register, next tcd address used in scatter-gather mode
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
uint8_t RESERVED[16]
    Aligned 64 bytes
struct __edma4_core_tcd
    #include <fsl_edma_core.h> edma4 core TCD struture definition

```

Public Members

```

__IO uint32_t SADDR
    SADDR register, used to save source address
__IO uint16_t SOFF
    SOFF register, save offset bytes every transfer
__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register

```

```
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
struct _edma_core_tcd
    #include <fsl_edma_core.h> edma core TCD struture definition
union MP_REGS
```

Public Members

```
struct _edma_core_mp EDMA5_REG
struct EDMA5_REG
```

Public Members

```
__IO uint32_t MP_INT_LOW
    Channel Control and Status, array offset: 0x10008, array step: 0x10000
__IO uint32_t MP_INT_HIGH
    Channel Control and Status, array offset: 0x1000C, array step: 0x10000
__IO uint32_t MP_HRS_LOW
    Channel Control and Status, array offset: 0x10010, array step: 0x10000
__IO uint32_t MP_HRS_HIGH
    Channel Control and Status, array offset: 0x10014, array step: 0x10000
__IO uint32_t MP_STOPCH
    Channel Control and Status, array offset: 0x10020, array step: 0x10000
__IO uint32_t MP_SSR_LOW
    Channel Control and Status, array offset: 0x10030, array step: 0x10000
__IO uint32_t MP_SSR_HIGH
    Channel Control and Status, array offset: 0x10034, array step: 0x10000
__IO uint32_t CH_GRPRI [64]
    Channel Control and Status, array offset: 0x10100, array step: 0x10000
__IO uint32_t CH_MUX [64]
    Channel Control and Status, array offset: 0x10200, array step: 0x10000
__IO uint32_t CH_PROT [64]
    Channel Control and Status, array offset: 0x10400, array step: 0x10000
union CH_REGS
```

Public Members

struct *_edma_core_channel* EDMA5_REG

struct *_edma_core_channel* EDMA4_REG

struct EDMA5_REG

Public Members

__IO uint32_t CH_MATTR

Memory Attributes Register, array offset: 0x10018, array step: 0x8000

struct EDMA4_REG

Public Members

__IO uint32_t CH_MUX

Channel Multiplexor Configuration, array offset: 0x10014, array step: 0x10000

__IO uint16_t CH_MATTR

Memory Attributes Register, array offset: 0x10018, array step: 0x8000

union TCD_REGS

Public Members

edma4_core_tcd_t edma4_tcd

2.8 eDMA soc Driver

FSL_EDMA_SOC_DRIVER_VERSION

Driver version 2.0.0.

FSL_EDMA_SOC_IP_DMA3

DMA IP version.

FSL_EDMA_SOC_IP_DMA4

EDMA_BASE_PTRS

DMA base table.

EDMA_CHN_IRQS

EDMA_CHANNEL_OFFSET

EDMA base address convert macro.

EDMA_CHANNEL_ARRAY_STEP(base)

2.9 EIM: error injection module

FSL_ERM_DRIVER_VERSION

Driver version.

void EIM_Init(EIM_Type *base)
EIM module initialization function.

Parameters

- base – EIM base address.

void EIM_Deinit(EIM_Type *base)
De-initializes the EIM.

2.10 ERM: error recording module

void ERM_Init(ERM_Type *base)
ERM module initialization function.

Parameters

- base – ERM base address.

void ERM_Deinit(ERM_Type *base)
De-initializes the ERM.

static inline void ERM_EnableInterrupts(ERM_Type *base, uint32_t channel, uint32_t mask)
ERM enable interrupts.

Parameters

- base – ERM peripheral base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to “_erm_interrupt_enable” enumeration.

static inline void ERM_DisableInterrupts(ERM_Type *base, uint32_t channel, uint32_t mask)
ERM module disable interrupts.

Parameters

- base – ERM base address.
- channel – memory channel.
- mask – single correction interrupt or non-correction interrupt enable to disable for one specific memory region. Refer to “_erm_interrupt_enable” enumeration.

static inline uint32_t ERM_GetInterruptStatus(ERM_Type *base, uint32_t channel)
Gets ERM interrupt flags.

Parameters

- base – ERM peripheral base address.

Returns

ERM event flags.

static inline void ERM_ClearInterruptStatus(ERM_Type *base, uint32_t channel, uint32_t mask)
ERM module clear interrupt status flag.

Parameters

- base – ERM base address.
- mask – event flag to clear. Refer to “_erm_interrupt_flag” enumeration.

uint32_t ERM_GetMemoryErrorAddr(ERM_Type *base, uint32_t channel)

ERM get memory error absolute address, which capturing the address of the last ECC event in Memory n.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

memory – error absolute address.

uint32_t ERM_GetSyndrome(ERM_Type *base, uint32_t channel)

ERM get syndrome, which identifies the pertinent bit position on a correctable, single-bit data inversion or a non-correctable, single-bit address inversion. The syndrome value does not provide any additional diagnostic information on non-correctable, multi-bit inversions.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

syndrome – value.

uint32_t ERM_GetErrorCount(ERM_Type *base, uint32_t channel)

ERM get error count, which records the count value of the number of correctable ECC error events for Memory n. Non-correctable errors are considered a serious fault, so the ERM does not provide any mechanism to count non-correctable errors. Only correctable errors are counted.

Parameters

- base – ERM base address.
- channel – memory channel.

Return values

error – count.

void ERM_ResetErrorCount(ERM_Type *base, uint32_t channel)

ERM reset error count.

Parameters

- base – ERM base address.
- channel – memory channel.

FSL_ERM_DRIVER_VERSION

Driver version.

ERM interrupt configuration structure, default settings all disabled, _erm_interrupt_enable. This structure contains the settings for all of the ERM interrupt configurations.

Values:

enumerator kERM_SingleCorrectionIntEnable

Single Correction Interrupt Notification enable.

enumerator kERM_NonCorrectableIntEnable

Non-Correction Interrupt Notification enable.

enumerator kERM_AllInterruptsEnable
All Interrupts enable

ERM interrupt status, `_erm_interrupt_flag`.

This provides constants for the ERM event status for use in the ERM functions.

Values:

enumerator kERM_SingleBitCorrectionIntFlag
Single-Bit Correction Event.

enumerator kERM_NonCorrectableErrorIntFlag
Non-Correctable Error Event.

enumerator kERM_AllIntsFlag
All Events.

2.11 EVTG: Event Generator Driver

FSL_EVTG_DRIVER_VERSION
EVTG driver version.
Version 2.0.2.

enum `_evtg_index`
EVTG instance index.

Values:

enumerator kEVTG_Index0
EVTG instance index 0.

enumerator kEVTG_Index1
EVTG instance index 1.

enumerator kEVTG_Index2
EVTG instance index 2.

enumerator kEVTG_Index3
EVTG instance index 3.

enum `_evtg_input_index`
EVTG input index.

Values:

enumerator kEVTG_InputA
EVTG input A.

enumerator kEVTG_InputB
EVTG input B.

enumerator kEVTG_InputC
EVTG input C.

enumerator kEVTG_InputD
EVTG input D.

enum _evtg_aoi_index

EVTG AOI index.

Values:

enumerator kEVTG_AOI0

EVTG AOI index 0.

enumerator kEVTG_AOI1

EVTG AOI index 1.

enum _evtg_aoi_product_term

EVTG AOI product term index.

Values:

enumerator kEVTG_ProductTerm0

EVTG AOI product term index 0.

enumerator kEVTG_ProductTerm1

EVTG AOI product term index 1.

enumerator kEVTG_ProductTerm2

EVTG AOI product term index 2.

enumerator kEVTG_ProductTerm3

EVTG AOI product term index 3.

enum _evtg_aoi_input_config

EVTG input configuration.

Values:

enumerator kEVTG_InputLogicZero

Force input in product term to a logical zero.

enumerator kEVTG_InputDirectPass

Pass input in product term.

enumerator kEVTG_InputComplement

Complement input in product term.

enumerator kEVTG_InputLogicOne

Force input in product term to a logical one.

enum _evtg_aoi_outfilter_count

EVTG AOI Output Filter Sample Count.

Values:

enumerator kEVTG_AOIOutFilterSampleCount3

EVTG AOI output filter sample count is 3.

enumerator kEVTG_AOIOutFilterSampleCount4

EVTG AOI output filter sample count is 4.

enumerator kEVTG_AOIOutFilterSampleCount5

EVTG AOI output filter sample count is 5.

enumerator kEVTG_AOIOutFilterSampleCount6

EVTG AOI output filter sample count is 6.

enumerator kEVTG_AOIOutFilterSampleCount7

EVTG AOI output filter sample count is 7.

enumerator kEVTG_AOIOutFilterSampleCount8

EVTG AOI output filter sample count is 8.

enumerator kEVTG_AOIOutFilterSampleCount9

EVTG AOI output filter sample count is 9.

enumerator kEVTG_AOIOutFilterSampleCount10

EVTG AOI output filter sample count is 10.

enum __evtg_outfdbk_override_input

EVTG output feedback override control mode. When FF is configured as JK-FF mode, need EVTG_OUTA feedback to EVTG input and replace one of the four inputs.

Values:

enumerator kEVTG_OutputOverrideInputA

Replace input A.

enumerator kEVTG_OutputOverrideInputB

Replace input B.

enumerator kEVTG_OutputOverrideInputC

Replace input C.

enumerator kEVTG_OutputOverrideInputD

Replace input D.

enum __evtg_flipflop_mode

EVTG flip flop mode configuration.

Values:

enumerator kEVTG_FFModeBypass

Bypass mode (default). In this mode, user can choose to enable or disable input sync logic and filter function.

enumerator kEVTG_FFModeRSTrigger

RS trigger mode. In this mode, user can choose to enable or disable input sync logic and filter function.

enumerator kEVTG_FFModeTFF

T-FF mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enumerator kEVTG_FFModeDFF

D-FF mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enumerator kEVTG_FFModeJKFF

JK-FF mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enumerator kEVTG_FFModeLatch

Latch mode. In this mode, input sync or filter has to be enabled to remove the possible glitch.

enum __evtg_flipflop_init_output

EVTG flip-flop initial value.

Values:

enumerator kEVTG_FFInitOut0

Configure the positive output of flip-flop as 0.

enumerator kEVTG_FFInitOut1

Configure the positive output of flip-flop as 1.

typedef enum *_evtg_index* evtg_index_t

EVTG instance index.

typedef enum *_evtg_input_index* evtg_input_index_t

EVTG input index.

typedef enum *_evtg_aoi_index* evtg_aoi_index_t

EVTG AOI index.

typedef enum *_evtg_aoi_product_term* evtg_aoi_product_term_t

EVTG AOI product term index.

typedef enum *_evtg_aoi_input_config* evtg_aoi_input_config_t

EVTG input configuration.

typedef enum *_evtg_aoi_outfilter_count* evtg_aoi_outfilter_count_t

EVTG AOI Output Filter Sample Count.

typedef enum *_evtg_outfdbk_override_input* evtg_outfdbk_override_input_t

EVTG output feedback override control mode. When FF is configured as JK-FF mode, need EVTG_OUTA feedback to EVTG input and replace one of the four inputs.

typedef enum *_evtg_flipflop_mode* evtg_flipflop_mode_t

EVTG flip flop mode configuration.

typedef enum *_evtg_flipflop_init_output* evtg_flipflop_init_output_t

EVTG flip-flop initial value.

typedef struct *_evtg_aoi_outfilter_config* evtg_aoi_outfilter_config_t

The structure for configuring an AOI output filter sample.

AOI output filter sample count represent the number of consecutive samples that must agree prior to the AOI output filter accepting an transition. AOI output filter sample period represent the sampling period (in IP bus clock cycles) of the AOI output signals. Each AOI output is sampled multiple times at the rate specified by this period.

For the modes with Filter function enabled, filter delay is “(FILT_CNT + 3) x FILT_PER + 2”.

typedef struct *_evtg_aoi_product_term_config* evtg_aoi_product_term_config_t

The structure for configuring an AOI product term.

typedef struct *_evtg_aoi_config* evtg_aoi_config_t

EVTG AOI configuration structure.

typedef struct *_evtg_config* evtg_config_t

EVTG configuration covering all configurable fields.

struct *_evtg_aoi_outfilter_config*

#include <fsl_evtg.h> The structure for configuring an AOI output filter sample.

AOI output filter sample count represent the number of consecutive samples that must agree prior to the AOI output filter accepting an transition. AOI output filter sample period represent the sampling period (in IP bus clock cycles) of the AOI output signals. Each AOI output is sampled multiple times at the rate specified by this period.

For the modes with Filter function enabled, filter delay is “(FILT_CNT + 3) x FILT_PER + 2”.

Public Members

evtg_aoi_outfilter_count_t sampleCount

EVTG AOI output filter sample count. refer to *evtg_aoi_outfilter_count_t*.

uint8_t samplePeriod

EVTG AOI output filter sample period, within 0~255. If sample period value is 0x00 (default), then the input filter is bypassed.

struct *_evtg_aoi_product_term_config*

#include <fsl_evtg.h> The structure for configuring an AOI product term.

Public Members

evtg_aoi_input_config_t aInput

Input A configuration.

evtg_aoi_input_config_t bInput

Input B configuration.

evtg_aoi_input_config_t cInput

Input C configuration.

evtg_aoi_input_config_t dInput

Input D configuration.

struct *_evtg_aoi_config*

#include <fsl_evtg.h> EVTG AOI configuration structure.

Public Members

evtg_aoi_outfilter_config_t aoiOutFilterConfig

EVTG AOI output filter sample configuration structure.

evtg_aoi_product_term_config_t productTerm0

Configure AOI product term0.

evtg_aoi_product_term_config_t productTerm1

Configure AOI product term1.

evtg_aoi_product_term_config_t productTerm2

Configure AOI product term2.

evtg_aoi_product_term_config_t productTerm3

Configure AOI product term3.

struct *_evtg_config*

#include <fsl_evtg.h> EVTG configuration covering all configurable fields.

Public Members

bool enableInputASync

Enable/Disable EVTG A input synchronous with bus clk.

bool enableInputBSync

Enable/Disable EVTG B input synchronous with bus clk.

bool enableInputCSync

Enable/Disable EVTG C input synchronous with bus clk.

bool enableInputDSync

Enable/Disable EVTG D input synchronous with bus clk.

evtg_outfdbk_override_input_t outfdbkOverrideinput

EVTG output feedback to EVTG input and replace one of the four inputs.

evtg_flipflop_mode_t flipflopMode

Flip-Flop can be configured as one of Bypass mode, RS trigger mode, T-FF mode, D-FF mode, JK-FF mode, Latch mode.

bool enableFlipflopInitOutput

Flip-flop initial output value enable/disable.

evtg_flipflop_init_output_t flipflopInitOutputValue

Flip-flop initial output value configuration.

bool enableForceBypassFlipFlopAOI0

Enable/Disable force bypass Flip-Flop and route the AOI_0(Filter_0) value directly to EVTG_OUTA

bool enableForceBypassFlipFlopAOI1

Enable/Disable force bypass Flip-Flop and route the AOI_1(Filter_1) value directly to EVTG_OUTB

evtg_aoi_config_t aoi0Config

Configure EVTG AOI0.

evtg_aoi_config_t aoi1Config

Configure EVTG AOI1.

2.12 EVTG Driver

void EVTG_Init(EVTG_Type *base, *evtg_index_t* evtgIndex, *evtg_config_t* *psConfig)

Initialize EVTG with a user configuration structure.

Parameters

- base – EVTG base address.
- evtgIndex – EVTG instance index.
- psConfig – EVTG initial configuration structure pointer.

void EVTG_GetDefaultConfig(*evtg_config_t* *psConfig, *evtg_flipflop_mode_t* flipflopMode)

Loads default values to the EVTG configuration structure.

The purpose of this API is to initialize the configuration structure to default value for EVTG_Init() to use. The Flip-Flop can be configured as Bypass mode, RS trigger mode, T-FF mode, D-FF mode, JK-FF mode, Latch mode. Please check RM INTC chapter for more details.

Parameters

- psConfig – EVTG initial configuration structure pointer.
- flipflopMode – EVTG flip flop mode. see @ ref _evtg_flipflop_mode

static inline void EVTG_ForceFlipflopInitOutput(EVTG_Type *base, *evtg_index_t* evtgIndex, *evtg_flipflop_init_output_t* flipflopInitOutputValue)

Force Flip-flop initial output value to be presented on flip-flop positive output.

Parameters

- base – EVTG base address.
- evtgIndex – EVTG instance index.
- flipflopInitOutputValue – EVTG flip-flop initial output control. see `evtg_flipflop_init_output_t`

```
static inline void EVTG_SetProductTermInput(EVTG_Type *base, evtg_index_t evtgIndex,
                                           evtg_aoi_index_t aoiIndex,
                                           evtg_aoi_product_term_t productTerm,
                                           evtg_input_index_t inputIndex,
                                           evtg_aoi_input_config_t input)
```

Configure each input value of AOI product term. Each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. Adapt to some simple aoi expressions.

Parameters

- base – EVTG base address.
- evtgIndex – EVTG instance index.
- aoiIndex – EVTG AOI index. see enum ref `evtg_aoi_index_t`
- productTerm – EVTG product term index.
- inputIndex – EVTG input index.
- input – EVTG input configuration with enum `evtg_aoi_input_config_t`.

```
void EVTG_ConfigAOIProductTerm(EVTG_Type *base, evtg_index_t evtgIndex, evtg_aoi_index_t
                               aoiIndex, evtg_aoi_product_term_t productTerm,
                               evtg_aoi_product_term_config_t *psProductTermConfig)
```

Configure AOI product term by initializing the product term configuration structure.

Parameters

- base – EVTG base address.
- evtgIndex – EVTG instance index.
- aoiIndex – EVTG AOI index. see enum `evtg_aoi_index_t`
- productTerm – EVTG AOI product term index.
- psProductTermConfig – Pointer to EVTG product term configuration structure. see ref `_evtg_aoi_product_term_config`

2.13 EWM: External Watchdog Monitor Driver

```
void EWM_Init(EWM_Type *base, const ewm_config_t *config)
```

Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

[ewm_config_t](#)

Parameters

- config – Pointer to the EWM configuration structure.

static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined
 - kEWM_InterruptEnable

static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to disable The parameter can be combination of the following source if defined
 - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;  
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

`_ewm_status_flags_t`

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)

Services the EWM.

This function resets the EWM counter to zero.

Parameters

- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION

EWM driver version 2.0.4.

enum _ewm_interrupt_enable_t

EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable

Enable the EWM to generate an interrupt

enum _ewm_status_flags_t

EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator kEWM_RunningFlag

Running flag, set when EWM is enabled

typedef struct _ewm_config ewm_config_t

Describes EWM clock source.

Data structure for EWM configuration.

This structure is used to configure the EWM.

struct _ewm_config

#include <fsl_ewm.h> Describes EWM clock source.

Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

`bool enableEwm`
Enable EWM module

`bool enableEwmInput`
Enable EWM_in input

`bool setInputAssertLogic`
EWM_in signal assertion state

`bool enableInterrupt`
Enable EWM interrupt

`uint8_t compareLowValue`
Compare low-register value

`uint8_t compareHighValue`
Compare high-register value

2.14 FGPIO Driver

2.15 FlexCAN: Flex Controller Area Network Driver

2.16 FlexCAN Driver

`bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)`
Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN_FDInit to enable CAN FD mode.

Parameters

- `base` – FlexCAN peripheral base address.

Returns

return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

`uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)`
Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080. For 32 bytes payload frame, the second mailbox is $((1/12)*512 + 1\%12*40)/4 = 10$, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- `base` – FlexCAN peripheral base address.

- mbIdx – Mailbox index.

Returns

Mailbox address offset in word.

`status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)`

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout
when wait for Freeze Mode Acknowledge

`status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)`

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout
when wait for Freeze Mode Acknowledge

`uint32_t FLEXCAN_GetInstance(CAN_Type *base)`

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN instance.

`bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t
sourceClock_Hz, flexcan_timing_config_t
*pTimingConfig)`

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

`void FLEXCAN_Init(CAN_Type *base, const flexcan_config_t *pConfig, uint32_t sourceClock_Hz)`

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.


```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.enableDoze    = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.timingConfig = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 4000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

```
bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t
                                             bitRateFD, uint32_t sourceClock_Hz,
                                             flexcan_timing_config_t *pTimingConfig)
```

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.
- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

```
void FLEXCAN_FDInit(CAN_Type *base, const flexcan_config_t *pConfig, uint32_t
                   sourceClock_Hz, flexcan_mb_size_t dataSize, bool brs)
```

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc      = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate     = 1000000U;
flexcanConfig.bitRateFD   = 2000000U;
flexcanConfig.maxMbNum    = 16;
flexcanConfig.enableLoopBack = false;
flexcanConfig.enableSelfWakeup = false;
flexcanConfig.enableIndividMask = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
```

(continues on next page)

(continued from previous page)

```
flexcanConfig.enableDoze      = false;
flexcanConfig.timingConfig    = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 8000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

void FLEXCAN_Deinit(CAN_Type *base)

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN_GetDefaultConfig(*flexcan_config_t* *pConfig)

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. `flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0;` `flexcanConfig->bitRate = 1000000U;` `flexcanConfig->bitRateFD = 2000000U;` `flexcanConfig->maxMbNum = 16;` `flexcanConfig->enableLoopBack = false;` `flexcanConfig->enableSelfWakeup = false;` `flexcanConfig->enableIndividMask = false;` `flexcanConfig->disableSelfReception = false;` `flexcanConfig->enableListenOnlyMode = false;` `flexcanConfig->enableDoze = false;` `flexcanConfig->enablePretendedNetworking = false;` `flexcanConfig->enableMemoryErrorControl = true;` `flexcanConfig->enableNonCorrectableErrorEnterFreeze = true;` `flexcanConfig->enableTransceiverDelayMeasure = true;` `flexcanConfig->enableRemoteRequestFrameStored = true;` `flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian;` `flexcanConfig.timingConfig = timingConfig;`

Parameters

- pConfig – Pointer to the FlexCAN configuration structure.

void FLEXCAN_SetTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetBitRate() instead.

Note: Calling FLEXCAN_SetTimingConfig() overrides the bit rate set in FLEXCAN_Init() or FLEXCAN_SetBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on FLEXCAN_CalculateImprovedTimingValues() API calculated timing values.

Note: Calling FLEXCAN_SetBitRate() overrides the bit rate set in FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRate_Bps – Bit rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only Nominal phase) successfully.

void FLEXCAN_SetFDTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetFDBitRate() to set both Nominal/Data bit Rate instead.

Note: Calling FLEXCAN_SetFDTimingConfig() overrides the data phase bit rate set in FLEXCAN_FDInit()/FLEXCAN_SetFDBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRateN_Bps, uint32_t bitRateD_Bps)

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on FLEXCAN_FDCalculateImprovedTimingValues() API calculated timing values.

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRateN_Bps – Nominal bit Rate in Bps.
- bitRateD_Bps – Data bit Rate in Bps.

Returns

kStatus_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN_SetTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
 - true: Enable Tx Message Buffer.
 - false: Disable Tx Message Buffer.

void FLEXCAN_SetRxMbConfig(CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbEmpty, kFLEXCAN_RxMbFull or kFLEXCAN_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.

- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
 - true: Enable Rx Message Buffer.
 - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
 - true: Enable Tx Message Buffer.
 - false: Disable Tx Message Buffer.

void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
 - true: Enable Rx Message Buffer.
 - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

```
void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const
                                         flexcan_frame_t *pFrame)
```

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

```
void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const flexcan_rx_fifo_config_t *pRxFifoConfig,
                             bool enable)
```

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note: Legacy Rx FIFO only can receive classic CAN message.

Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Legacy Rx FIFO.
 - true: Enable Legacy Rx FIFO.
 - false: Disable Legacy Rx FIFO.

```
void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const
                                       flexcan_enhanced_rx_fifo_config_t *pConfig, bool
                                       enable)
```

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

Note: Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
 - true: Enable Enhanced Rx FIFO.
 - false: Disable Enhanced Rx FIFO.

```
void FLEXCAN_SetPNConfig(CAN_Type *base, const flexcan_pn_config_t *pConfig)
```

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

```
static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)
```

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators `_flexcan_flags`. To check the specific status, compare the return value with enumerators in `_flexcan_flags`.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the `_flexcan_flags`.

```
static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of `_flexcan_flags`.

```
static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)
```

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

```
static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)
```

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

```
static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)
```

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

```
static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)
```

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

```
void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base,  
                                         flexcan_memory_error_report_status_t  
                                         *errorStatus)
```

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

```
static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)
```

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of received wake up messages.

```
static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)
```

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

Parameters

- `base` – FlexCAN peripheral base address.
- `mask` – The interrupts to enable. Logical OR of `_flexcan_interrupt_enable`.

static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see `_flexcan_interrupt_enable`.

Parameters

- `base` – FlexCAN peripheral base address.
- `mask` – The interrupts to disable. Logical OR of `_flexcan_interrupt_enable`.

static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

Parameters

- `base` – FlexCAN peripheral base address.
- `mask` – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- `base` – FlexCAN peripheral base address.
- `mask` – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

Parameters

- `base` – FlexCAN peripheral base address.
- `mask` – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- `base` – FlexCAN peripheral base address.
- `mask` – The ORed FlexCAN Message Buffer mask.

`void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)`

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

- `base` – FlexCAN peripheral base address.
- `enable` – true to enable, false to disable.

`static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)`

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

- `base` – FlexCAN peripheral base address.

Returns

FlexCAN Rx FIFO Head address.

`static inline status_t FLEXCAN_Enable(CAN_Type *base, bool enable)`

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

Parameters

- `base` – FlexCAN base pointer.
- `enable` – true to enable, false to disable.

Returns

`kStatus_Success` Enable FlexCAN module successful
`kStatus_Timeout` Timeout when wait for Low-Power Mode Acknowledge

`status_t FLEXCAN_WriteTxMb(CAN_Type *base, uint8_t mbIdx, const flexcan_frame_t *pTxFrame)`

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.
- `mbIdx` – The FlexCAN Message Buffer index.
- `pTxFrame` – Pointer to CAN message frame to be sent.

Return values

- `kStatus_Success` – - Write Tx Message Buffer Successfully.
- `kStatus_Fail` – - Tx Message Buffer is currently in use.

`status_t FLEXCAN_ReadRxMb(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pRxFrame)`

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- `base` – FlexCAN peripheral base address.

- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – Rx Message Buffer is empty.

status_t FLEXCAN_WriteFDTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_fd_frame_t* *pTxFrame)

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – Write Tx Message Buffer Successfully.
- kStatus_Fail – Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadFDRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – Rx Message Buffer is empty.

status_t FLEXCAN_ReadRxFifo(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- `kStatus_Success` -- Read Message from Rx FIFO successfully.
- `kStatus_Fail` -- Rx FIFO is not enabled.

`status_t` FLEXCAN_ReadEnhancedRxFifo(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

- `base` – FlexCAN peripheral base address.
- `pRxFrame` – Pointer to CAN FD message frame structure for reception.

Return values

- `kStatus_Success` -- Read Message from Rx FIFO successfully.
- `kStatus_Fail` -- Rx FIFO is not enabled.

`status_t` FLEXCAN_ReadPNWakeUpMB(CAN_Type *base, `uint8_t` mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

Parameters

- `base` – FlexCAN peripheral base address.
- `pRxFrame` – Pointer to CAN message frame structure for reception.
- `mbIdx` – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

Return values

- `kStatus_Success` -- Read Message from Wake up Message Buffer successfully.
- `kStatus_Fail` -- Wake up Message Buffer has no valid content.

`status_t` FLEXCAN_TransferFDSendBlocking(CAN_Type *base, `uint8_t` mbIdx, *flexcan_fd_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- `base` – FlexCAN peripheral base pointer.
- `mbIdx` – The FlexCAN FD Message Buffer index.
- `pTxFrame` – Pointer to CAN FD message frame to be sent.

Return values

- `kStatus_Success` -- Write Tx Message Buffer Successfully.
- `kStatus_Fail` -- Tx Message Buffer is currently in use.
- `kStatus_Timeout` -- Failed to send frames within specific time.

status_t FLEXCAN_TransferFDReceiveBlocking(CAN_Type *base, uint8_t mbIdx,
flexcan_fd_frame_t *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – Rx Message Buffer is empty.
- kStatus_Timeout – Failed to receive frames within specific time.

status_t FLEXCAN_TransferFDSendNonBlocking(CAN_Type *base, flexcan_handle_t *handle,
flexcan_mb_transfer_t *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferFDReceiveNonBlocking(CAN_Type *base, flexcan_handle_t *handle,
flexcan_mb_transfer_t *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Rx Message Buffer receiving process successfully.

- kStatus_FLEXCAN_RxBusy – Rx Message Buffer is in use.

void FLEXCAN_TransferFDAbortSend(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

void FLEXCAN_TransferFDAbortReceive(CAN_Type *base, flexcan_handle_t *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

status_t FLEXCAN_TransferSendBlocking(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – Write Tx Message Buffer Successfully.
- kStatus_Fail – Tx Message Buffer is currently in use.
- kStatus_Timeout – Failed to send frames within specific time.

status_t FLEXCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- `kStatus_Success` -- Rx Message Buffer is full and has been read successfully.
- `kStatus_FLEXCAN_RxOverflow` -- Rx Message Buffer is already overflowed and has been read successfully.
- `kStatus_Fail` -- Rx Message Buffer is empty.
- `kStatus_Timeout` -- Failed to receive frames within specific time.

`status_t` FLEXCAN_TransferReceiveFifoBlocking(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- `base` – FlexCAN peripheral base pointer.
- `pRxFrame` – Pointer to CAN message frame structure for reception.

Return values

- `kStatus_Success` -- Read Message from Rx FIFO successfully.
- `kStatus_Fail` -- Rx FIFO is not enabled.
- `kStatus_Timeout` -- Failed to receive frames within specific time.

`status_t` FLEXCAN_TransferReceiveEnhancedFifoBlocking(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- `base` – FlexCAN peripheral base pointer.
- `pRxFrame` – Pointer to CAN FD message frame structure for reception.

Return values

- `kStatus_Success` -- Read Message from Rx FIFO successfully.
- `kStatus_Fail` -- Rx FIFO is not enabled.
- `kStatus_Timeout` -- Failed to receive frames within specific time.

`void` FLEXCAN_TransferCreateHandle(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_transfer_callback_t* callback, void *userData)

Initializes the FlexCAN handle.

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

Parameters

- `base` – FlexCAN peripheral base address.
- `handle` – FlexCAN handle pointer.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

status_t FLEXCAN_TransferSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – Rx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the *flexcan_fifo_transfer_t*.

Return values

- kStatus_Success – Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – Rx FIFO is currently in use.


```
status_t FLEXCAN_TransferGetReceiveFifoCount(CAN_Type *base, flexcan_handle_t *handle,
                                             size_t *count)
```

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- *kStatus_InvalidArgument* – count is Invalid.
- *kStatus_Success* – Successfully return the count.

```
status_t FLEXCAN_TransferReceiveEnhancedFifoNonBlocking(CAN_Type *base, flexcan_handle_t
                                                         *handle, flexcan_fifo_transfer_t
                                                         *pFifoXfer)
```

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- *pFifoXfer* – FlexCAN Rx FIFO transfer structure. See the ref *flexcan_fifo_transfer_t*.

Return values

- *kStatus_Success* – Start Rx FIFO receiving process successfully.
- *kStatus_FLEXCAN_RxFifoBusy* – Rx FIFO is currently in use.

```
static inline status_t FLEXCAN_TransferGetReceiveEnhancedFifoCount(CAN_Type *base,
                                                                    flexcan_handle_t *handle,
                                                                    size_t *count)
```

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- *kStatus_InvalidArgument* – count is Invalid.
- *kStatus_Success* – Successfully return the count.

```
uint32_t FLEXCAN_GetTimeStamp(flexcan_handle_t *handle, uint8_t mbIdx)
```

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking

-FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -
FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

void FLEXCAN_TransferAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferAbortReceiveEnhancedFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Enhanced Rx FIFO process.

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_MbHandleIRQ(CAN_Type *base, flexcan_handle_t *handle, uint32_t startMbIdx,
                        uint32_t endMbIdx)
```

FlexCAN Message Buffer IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

```
void FLEXCAN_EhancedRxFifoHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)
```

FlexCAN Enhanced Rx FIFO IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_BusoffErrorHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)
```

FlexCAN Bus Off, Error and Warning IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_PNWakeUpHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)
```

FlexCAN Pretended Networking Wake-up IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
void FLEXCAN_MemoryErrorHandleIRQ(CAN_Type *base, flexcan_handle_t *handle)
```

FlexCAN Memory Error IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

```
FSL_FLEXCAN_DRIVER_VERSION
```

FlexCAN driver version.

FlexCAN transfer status.

Values:

enumerator kStatus_FLEXCAN_TxBusy

Tx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_TxIdle

Tx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_TxSwitchToRx

Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus_FLEXCAN_RxBusy

Rx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_RxIdle
Rx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_RxOverflow
Rx Message Buffer is Overflowed.

enumerator kStatus_FLEXCAN_RxFifoBusy
Rx Message FIFO is Busy.

enumerator kStatus_FLEXCAN_RxFifoIdle
Rx Message FIFO is Idle.

enumerator kStatus_FLEXCAN_RxFifoOverflow
Rx Message FIFO is overflowed.

enumerator kStatus_FLEXCAN_RxFifoWarning
Rx Message FIFO is almost overflowed.

enumerator kStatus_FLEXCAN_RxFifoDisabled
Rx Message FIFO is disabled during reading.

enumerator kStatus_FLEXCAN_ErrorStatus
FlexCAN Module Error and Status.

enumerator kStatus_FLEXCAN_WakeUp
FlexCAN is waken up from STOP mode.

enumerator kStatus_FLEXCAN_UnHandled
UnHadled Interrupt asserted.

enumerator kStatus_FLEXCAN_RxRemote
Rx Remote Message Received in Mail box.

enumerator kStatus_FLEXCAN_RxFifoUnderflow
Enhanced Rx Message FIFO is underflow.

enumerator kStatus_FLEXCAN_MemoryError
FlexCAN Memory Error.

enum _flexcan_frame_format
FlexCAN frame format.

Values:

enumerator kFLEXCAN_FrameFormatStandard
Standard frame format attribute.

enumerator kFLEXCAN_FrameFormatExtend
Extend frame format attribute.

enum _flexcan_frame_type
FlexCAN frame type.

Values:

enumerator kFLEXCAN_FrameTypeData
Data frame type attribute.

enumerator kFLEXCAN_FrameTypeRemote
Remote frame type attribute.

enum _flexcan_clock_source
FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Values:

enumerator kFLEXCAN_ClkSrcOsc

FlexCAN Protocol Engine clock from Oscillator.

enumerator kFLEXCAN_ClkSrcPeri

FlexCAN Protocol Engine clock from Peripheral Clock.

enumerator kFLEXCAN_ClkSrc0

FlexCAN Protocol Engine clock selected by user as SRC == 0.

enumerator kFLEXCAN_ClkSrc1

FlexCAN Protocol Engine clock selected by user as SRC == 1.

enum _flexcan_wake_up_source

FlexCAN wake up source.

Values:

enumerator kFLEXCAN_WakeupSrcUnfiltered

FlexCAN uses unfiltered Rx input to detect edge.

enumerator kFLEXCAN_WakeupSrcFiltered

FlexCAN uses filtered Rx input to detect edge.

enum _flexcan_rx_fifo_filter_type

FlexCAN Rx Fifo Filter type.

Values:

enumerator kFLEXCAN_RxFifoFilterTypeA

One full ID (standard and extended) per ID Filter element.

enumerator kFLEXCAN_RxFifoFilterTypeB

Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeC

Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeD

All frames rejected.

enum _flexcan_mb_size

FlexCAN Message Buffer Payload size.

Values:

enumerator kFLEXCAN_8BperMB

Selects 8 bytes per Message Buffer.

enumerator kFLEXCAN_16BperMB

Selects 16 bytes per Message Buffer.

enumerator kFLEXCAN_32BperMB

Selects 32 bytes per Message Buffer.

enumerator kFLEXCAN_64BperMB

Selects 64 bytes per Message Buffer.

enum _flexcan_fd_frame_length

FlexCAN CAN FD frame supporting data length (available DLC values).

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

Values:

enumerator kFLEXCAN_0BperFrame

Frame contains 0 valid data bytes.

enumerator kFLEXCAN_1BperFrame

Frame contains 1 valid data bytes.

enumerator kFLEXCAN_2BperFrame

Frame contains 2 valid data bytes.

enumerator kFLEXCAN_3BperFrame

Frame contains 3 valid data bytes.

enumerator kFLEXCAN_4BperFrame

Frame contains 4 valid data bytes.

enumerator kFLEXCAN_5BperFrame

Frame contains 5 valid data bytes.

enumerator kFLEXCAN_6BperFrame

Frame contains 6 valid data bytes.

enumerator kFLEXCAN_7BperFrame

Frame contains 7 valid data bytes.

enumerator kFLEXCAN_8BperFrame

Frame contains 8 valid data bytes.

enumerator kFLEXCAN_12BperFrame

Frame contains 12 valid data bytes.

enumerator kFLEXCAN_16BperFrame

Frame contains 16 valid data bytes.

enumerator kFLEXCAN_20BperFrame

Frame contains 20 valid data bytes.

enumerator kFLEXCAN_24BperFrame

Frame contains 24 valid data bytes.

enumerator kFLEXCAN_32BperFrame

Frame contains 32 valid data bytes.

enumerator kFLEXCAN_48BperFrame

Frame contains 48 valid data bytes.

enumerator kFLEXCAN_64BperFrame

Frame contains 64 valid data bytes.

enum _flexcan_efifo_dma_per_read_length

FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

Values:

enumerator kFLEXCAN_1WordPerRead
Transfer 1 32-bit words (CS).

enumerator kFLEXCAN_2WordPerRead
Transfer 2 32-bit words (CS + ID).

enumerator kFLEXCAN_3WordPerRead
Transfer 3 32-bit words (CS + ID + 1~4 bytes data).

enumerator kFLEXCAN_4WordPerRead
Transfer 4 32-bit words (CS + ID + 5~8 bytes data).

enumerator kFLEXCAN_5WordPerRead
Transfer 5 32-bit words (CS + ID + 9~12 bytes data).

enumerator kFLEXCAN_6WordPerRead
Transfer 6 32-bit words (CS + ID + 13~16 bytes data).

enumerator kFLEXCAN_7WordPerRead
Transfer 7 32-bit words (CS + ID + 17~20 bytes data).

enumerator kFLEXCAN_8WordPerRead
Transfer 8 32-bit words (CS + ID + 21~24 bytes data).

enumerator kFLEXCAN_9WordPerRead
Transfer 9 32-bit words (CS + ID + 25~28 bytes data).

enumerator kFLEXCAN_10WordPerRead
Transfer 10 32-bit words (CS + ID + 29~32 bytes data).

enumerator kFLEXCAN_11WordPerRead
Transfer 11 32-bit words (CS + ID + 33~36 bytes data).

enumerator kFLEXCAN_12WordPerRead
Transfer 12 32-bit words (CS + ID + 37~40 bytes data).

enumerator kFLEXCAN_13WordPerRead
Transfer 13 32-bit words (CS + ID + 41~44 bytes data).

enumerator kFLEXCAN_14WordPerRead
Transfer 14 32-bit words (CS + ID + 45~48 bytes data).

enumerator kFLEXCAN_15WordPerRead
Transfer 15 32-bit words (CS + ID + 49~52 bytes data).

enumerator kFLEXCAN_16WordPerRead
Transfer 16 32-bit words (CS + ID + 53~56 bytes data).

enumerator kFLEXCAN_17WordPerRead
Transfer 17 32-bit words (CS + ID + 57~60 bytes data).

enumerator kFLEXCAN_18WordPerRead
Transfer 18 32-bit words (CS + ID + 61~64 bytes data).

enumerator kFLEXCAN_19WordPerRead
Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT).

enum flexcan_rx_fifo_priority
FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

Values:

enumerator kFLEXCAN_RxFifoPrioLow

Matching process start from Rx Message Buffer first.

enumerator kFLEXCAN_RxFifoPrioHigh

Matching process start from Enhanced/Legacy Rx FIFO first.

enum _flexcan_interrupt_enable

FlexCAN interrupt enable enumerations.

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

Note: FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Values:

enumerator kFLEXCAN_BusOffInterruptEnable

Bus Off interrupt, use bit 15.

enumerator kFLEXCAN_ErrorInterruptEnable

CAN Error interrupt, use bit 14.

enumerator kFLEXCAN_TxWarningInterruptEnable

Tx Warning interrupt, use bit 11.

enumerator kFLEXCAN_RxWarningInterruptEnable

Rx Warning interrupt, use bit 10.

enumerator kFLEXCAN_WakeUpInterruptEnable

Self Wake Up interrupt, use bit 26.

enumerator kFLEXCAN_FDErrorInterruptEnable

CAN FD Error interrupt, use bit 31.

enumerator kFLEXCAN_PNMatchWakeUpInterruptEnable

PN Match Wake Up interrupt, use high word bit 17.

enumerator kFLEXCAN_PNTimeoutWakeUpInterruptEnable

PN Timeout Wake Up interrupt, use high word bit 16. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

enumerator kFLEXCAN_ERxFifoUnderflowInterruptEnable

Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

enumerator kFLEXCAN_ERxFifoOverflowInterruptEnable

Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

enumerator kFLEXCAN_ERxFifoWatermarkInterruptEnable

Enhanced Rx FIFO Data Available interrupt, use high word bit 28.

enumerator kFLEXCAN_ERxFifoDataAvlInterruptEnable

enumerator kFLEXCAN_HostAccessNCErrInterruptEnable

Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN_FlexCanAccessNCErrInterruptEnable

FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN_HostOrFlexCanCErrInterruptEnable

Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum _flexcan_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note: The CPU read action clears the bits corresponding to the FLEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using _flexcan_error_flags enumerations.

Values:

enumerator kFLEXCAN_ErrorOverrunFlag

Error Overrun Status.

enumerator kFLEXCAN_FDErrorIntFlag

CAN FD Error Interrupt Flag.

enumerator kFLEXCAN_BusoffDoneIntFlag

Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN_SynchFlag

CAN Synchronization Status.

enumerator kFLEXCAN_TxWarningIntFlag

Tx Warning Interrupt Flag.

enumerator kFLEXCAN_RxWarningIntFlag

Rx Warning Interrupt Flag.

enumerator kFLEXCAN_IdleFlag

FlexCAN In IDLE Status.

enumerator kFLEXCAN_FaultConfinementFlag

FlexCAN Fault Confinement State.

enumerator kFLEXCAN_TransmittingFlag

FlexCAN In Transmission Status.

enumerator kFLEXCAN_ReceivingFlag

FlexCAN In Reception Status.

enumerator kFLEXCAN_BusOffIntFlag

Bus Off Interrupt Flag.

enumerator kFLEXCAN_ErrorIntFlag

CAN Error Interrupt Flag.

enumerator kFLEXCAN_WakeUpIntFlag

Self Wake-Up Interrupt Flag.

enumerator kFLEXCAN_ErrorFlag

enumerator kFLEXCAN_PNMatchIntFlag

PN Matching Event Interrupt Flag.

enumerator kFLEXCAN_PNTimeoutIntFlag

PN Timeout Event Interrupt Flag.

enumerator kFLEXCAN_ERxFifoUnderflowIntFlag

Enhanced Rx FIFO underflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoOverflowIntFlag

Enhanced Rx FIFO overflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoWatermarkIntFlag

Enhanced Rx FIFO watermark Interrupt Flag.

enumerator kFLEXCAN_ERxFifoDataAvlIntFlag

Enhanced Rx FIFO data available Interrupt Flag.

enumerator kFLEXCAN_ERxFifoEmptyFlag

Enhanced Rx FIFO empty status.

enumerator kFLEXCAN_ERxFifoFullFlag

Enhanced Rx FIFO full status.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorIntFlag

Host Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag

FlexCAN Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_CorrectableErrorIntFlag

Correctable Error Interrupt Flag.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag

Host Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag

FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_CorrectableErrorOverrunFlag

Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_AllMemoryErrorIntFlag

All Memory Error Interrupt Flags.

enumerator kFLEXCAN_AllMemoryErrorFlag

All Memory Error Flags.

enum _flexcan_error_flags

FlexCAN error status flags.

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in _flexcan_flags enumerations to determine which error is generated.

Values:

enumerator kFLEXCAN_FDStuffingError

Stuffing Error.

enumerator kFLEXCAN_FDFormError

Form Error.

enumerator kFLEXCAN_FDCrcError

Cyclic Redundancy Check Error.

enumerator kFLEXCAN_FDBit0Error

Unable to send dominant bit.

enumerator kFLEXCAN_FDBit1Error

Unable to send recessive bit.

enumerator kFLEXCAN_TxErrorWarningFlag

Tx Error Warning Status.

enumerator kFLEXCAN_RxErrorWarningFlag

Rx Error Warning Status.

enumerator kFLEXCAN_StuffingError

Stuffing Error.

enumerator kFLEXCAN_FormError

Form Error.

enumerator kFLEXCAN_CrcError

Cyclic Redundancy Check Error.

enumerator kFLEXCAN_AckError

Received no ACK on transmission.

enumerator kFLEXCAN_Bit0Error

Unable to send dominant bit.

enumerator kFLEXCAN_Bit1Error

Unable to send recessive bit.

FlexCAN Legacy Rx FIFO status flags.

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Values:

enumerator kFLEXCAN_RxFifoOverflowFlag

Rx FIFO overflow flag.

enumerator kFLEXCAN_RxFifoWarningFlag

Rx FIFO almost full flag.

enumerator kFLEXCAN_RxFifoFrameAvlFlag

Frames available in Rx FIFO flag.

enum _flexcan_memory_error_type

FlexCAN Memory Error Type.

Values:

enumerator kFLEXCAN_CorrectableError

The memory error is correctable which means on bit error.

enumerator kFLEXCAN_NonCorrectableError

The memory error is non-correctable which means two bit errors.

enum _flexcan_memory_access_type

FlexCAN Memory Access Type.

Values:

enumerator kFLEXCAN_MoveOutFlexCanAccess

The memory error was detected during move-out FlexCAN access.

enumerator kFLEXCAN_MoveInAccess

The memory error was detected during move-in FlexCAN access.

enumerator kFLEXCAN_TxArbitrationAccess

The memory error was detected during Tx Arbitration FlexCAN access.

enumerator kFLEXCAN_RxMatchingAccess

The memory error was detected during Rx Matching FlexCAN access.

enumerator kFLEXCAN_MoveOutHostAccess

The memory error was detected during Rx Matching Host (CPU) access.

enum _flexcan_byte_error_syndrome

FlexCAN Memory Error Byte Syndrome.

Values:

enumerator kFLEXCAN_NoError

No bit error in this byte.

enumerator kFLEXCAN_ParityBits0Error

Parity bit 0 error in this byte.

enumerator kFLEXCAN_ParityBits1Error

Parity bit 1 error in this byte.

enumerator kFLEXCAN_ParityBits2Error

Parity bit 2 error in this byte.

enumerator kFLEXCAN_ParityBits3Error

Parity bit 3 error in this byte.

enumerator kFLEXCAN_ParityBits4Error

Parity bit 4 error in this byte.

enumerator kFLEXCAN_DataBits0Error

Data bit 0 error in this byte.

enumerator kFLEXCAN_DataBits1Error

Data bit 1 error in this byte.

enumerator kFLEXCAN_DataBits2Error

Data bit 2 error in this byte.

enumerator kFLEXCAN_DataBits3Error

Data bit 3 error in this byte.

enumerator kFLEXCAN_DataBits4Error

Data bit 4 error in this byte.

enumerator kFLEXCAN_DataBits5Error

Data bit 5 error in this byte.

enumerator kFLEXCAN_DataBits6Error

Data bit 6 error in this byte.

enumerator kFLEXCAN_DataBits7Error

Data bit 7 error in this byte.

enumerator kFLEXCAN_AllZeroError

All-zeros non-correctable error in this byte.

enumerator kFLEXCAN_AllOneError

All-ones non-correctable error in this byte.

enumerator kFLEXCAN_NonCorrectableErrors

Non-correctable error in this byte.

enum _flexcan_pn_match_source

FlexCAN Pretended Networking match source selection.

Values:

enumerator kFLEXCAN_PNMatSrcID

Message match with ID filtering.

enumerator kFLEXCAN_PNMatSrcIDAndData

Message match with ID filtering and payload filtering.

enum _flexcan_pn_match_mode

FlexCAN Pretended Networking mode match type.

Values:

enumerator kFLEXCAN_PNMatModeEqual

Match upon ID/Payload contents against an exact target value.

enumerator kFLEXCAN_PNMatModeGreater

Match upon an ID/Payload value greater than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeSmaller

Match upon an ID/Payload value smaller than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeRange

Match upon an ID/Payload value inside a range, greater than or equal to a specified lower limit, and smaller than or equal to a specified upper limit

typedef enum _flexcan_frame_format flexcan_frame_format_t

FlexCAN frame format.

typedef enum _flexcan_frame_type flexcan_frame_type_t

FlexCAN frame type.

typedef enum _flexcan_clock_source flexcan_clock_source_t

FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

typedef enum _flexcan_wake_up_source flexcan_wake_up_source_t

FlexCAN wake up source.

typedef enum _flexcan_rx_fifo_filter_type flexcan_rx_fifo_filter_type_t

FlexCAN Rx Fifo Filter type.

typedef enum _flexcan_mb_size flexcan_mb_size_t

FlexCAN Message Buffer Payload size.

typedef enum _flexcan_efifo_dma_per_read_length flexcan_efifo_dma_per_read_length_t

FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

typedef enum _flexcan_rx_fifo_priority flexcan_rx_fifo_priority_t

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

typedef enum *_flexcan_memory_error_type* flexcan_memory_error_type_t
FlexCAN Memory Error Type.

typedef enum *_flexcan_memory_access_type* flexcan_memory_access_type_t
FlexCAN Memory Access Type.

typedef enum *_flexcan_byte_error_syndrome* flexcan_byte_error_syndrome_t
FlexCAN Memory Error Byte Syndrome.

typedef struct *_flexcan_memory_error_report_status* flexcan_memory_error_report_status_t
FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN_GetMemoryErrorReportStatus() function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

typedef struct *_flexcan_frame* flexcan_frame_t
FlexCAN message frame structure.

typedef struct *_flexcan_fd_frame* flexcan_fd_frame_t
CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see *_flexcan_fd_frame_length*.

typedef struct *_flexcan_timing_config* flexcan_timing_config_t
FlexCAN protocol timing characteristic configuration structure.

typedef struct *_flexcan_config* flexcan_config_t
FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

typedef struct *_flexcan_rx_mb_config* flexcan_rx_mb_config_t
FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

typedef enum *_flexcan_pn_match_source* flexcan_pn_match_source_t
FlexCAN Pretended Networking match source selection.

typedef enum *_flexcan_pn_match_mode* flexcan_pn_match_mode_t
FlexCAN Pretended Networking mode match type.

typedef struct *_flexcan_pn_config* flexcan_pn_config_t
FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

typedef struct *_flexcan_rx_fifo_config* flexcan_rx_fifo_config_t
FlexCAN Legacy Rx FIFO configuration structure.

`typedef struct flexcan_enhanced_rx_fifo_std_id_filter flexcan_enhanced_rx_fifo_std_id_filter_t`
FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

`typedef struct flexcan_enhanced_rx_fifo_ext_id_filter flexcan_enhanced_rx_fifo_ext_id_filter_t`
FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

`typedef struct flexcan_enhanced_rx_fifo_config flexcan_enhanced_rx_fifo_config_t`
FlexCAN Enhanced Rx FIFO configuration structure.

`typedef struct flexcan_mb_transfer flexcan_mb_transfer_t`
FlexCAN Message Buffer transfer.

`typedef struct flexcan_fifo_transfer flexcan_fifo_transfer_t`
FlexCAN Rx FIFO transfer.

`typedef struct flexcan_handle flexcan_handle_t`
FlexCAN handle structure definition.

`typedef void (*flexcan_transfer_callback_t)(CAN_Type *base, flexcan_handle_t *handle, status_t status, uint64_t result, void *userData)`

`FLEXCAN_WAIT_TIMEOUT`

`FLEXCAN_POLLING_TIMEOUT`
Max loops to wait for polling transfer.

`FLEXCAN_MODULE_TIMEOUT`
Max loops to wait for FlexCAN register access complete.

`DLC_LENGTH_DECODE(dlc)`
FlexCAN frame length helper macro.

`FLEXCAN_ID_STD(id)`
FlexCAN Frame ID helper macro.
Standard Frame ID helper macro.

`FLEXCAN_ID_EXT(id)`
Extend Frame ID helper macro.

`FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)`
FlexCAN Rx Message Buffer Mask helper macro.
Standard Rx Message Buffer Mask helper macro.

`FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)`
Extend Rx Message Buffer Mask helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)`
FlexCAN Legacy Rx FIFO Mask helper macro.
Standard Rx FIFO Mask helper macro Type A helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)`
Standard Rx FIFO Mask helper macro Type B upper part helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)`
Standard Rx FIFO Mask helper macro Type B lower part helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id)`
Standard Rx FIFO Mask helper macro Type C upper part helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id)`
Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id)`

Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

`FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)`

Standard Rx FIFO Mask helper macro Type C lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)`

Extend Rx FIFO Mask helper macro Type A helper macro.

`FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)`

Extend Rx FIFO Mask helper macro Type B upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)`

Extend Rx FIFO Mask helper macro Type B lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)`

Extend Rx FIFO Mask helper macro Type C upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)`

Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)`

Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)`

Extend Rx FIFO Mask helper macro Type C lower part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)`

FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)`

Standard Rx FIFO Filter helper macro Type B upper part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)`

Standard Rx FIFO Filter helper macro Type B lower part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)`

Standard Rx FIFO Filter helper macro Type C upper part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)`

Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)`

Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

`FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)`

Standard Rx FIFO Filter helper macro Type C lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type A helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type B upper part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)`

Extend Rx FIFO Filter helper macro Type B lower part helper macro.

`FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)`

Extend Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)

Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)

Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id)

Extend Rx FIFO Filter helper macro Type C lower part helper macro.

ENHANCED_RX_FIFO_FSCH(x)

FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.

RTR_STD_HIGH(x)

RTR_STD_LOW(x)

RTR_EXT(x)

ID_STD_LOW(id)

ID_STD_HIGH(id)

ID_EXT(id)

FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask)

Standard ID filter element with filter + mask scheme.

FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE(id_upper, rtr, id_lower,
rtr_mask)

Standard ID filter element with filter range.

FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS(id1, rtr1, id2, rtr2)

Standard ID filter element with two filters without masks.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW(id, rtr)

Extended ID filter element with filter + mask scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH(id_mask, rtr_mask)

Extended ID filter element with filter + mask scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW(id_upper, rtr)

Extended ID filter element with range scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH(id_lower, rtr_mask)

Extended ID filter element with range scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW(id2, rtr2)

Extended ID filter element with two filters without masks low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH(id1, rtr1)

Extended ID filter element with two filters without masks high word.

FLEXCAN_PN_STD_MASK(id, rtr)

FlexCAN Pretended Networking ID Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_PN_EXT_MASK(id, rtr)

Extend Rx Message Buffer Mask helper macro.

FLEXCAN_PN_INT_MASK(x)

FlexCAN interrupt/status flag helper macro.

FLEXCAN_PN_INT_UNMASK(x)
FLEXCAN_PN_STATUS_MASK(x)
FLEXCAN_PN_STATUS_UNMASK(x)
FLEXCAN_EFIFO_INT_MASK(x)
FLEXCAN_EFIFO_INT_UNMASK(x)
FLEXCAN_EFIFO_STATUS_MASK(x)
FLEXCAN_EFIFO_STATUS_UNMASK(x)
FLEXCAN_MECR_INT_MASK(x)
FLEXCAN_MECR_INT_UNMASK(x)
FLEXCAN_MECR_STATUS_MASK(x)
FLEXCAN_MECR_STATUS_UNMASK(x)
FLEXCAN_ERROR_AND_STATUS_INT_FLAG
FLEXCAN_PNWAKE_UP_FLAG
FLEXCAN_WAKE_UP_FLAG
FLEXCAN_MEMORY_ERROR_INT_FLAG
FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG

FlexCAN Enhanced Rx FIFO base address helper macro.

E_RX_FIFO(base)

FLEXCAN_CALLBACK(x)

FlexCAN transfer callback function.

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_FLEXCAN_ErrorStatus`, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

struct flexcan_memory_error_report_status

#include <fsl_flexcan.h> FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of `FLEXCAN_GetMemoryErrorReportStatus()` function. And user can use `FLEXCAN_GetMemoryErrorReportStatus` to get the status of the last memory error access.

Public Members

flexcan_memory_error_type_t errorType

The type of memory error that giving rise to the report.

flexcan_memory_access_type_t accessType

The type of memory access that giving rise to the memory error.

uint16_t accessAddress

The address where memory error detected.

uint32_t errorData

The raw data word read from memory with error.

struct _flexcan_frame

#include <fsl_flexcan.h> FlexCAN message frame structure.

struct _flexcan_fd_frame

#include <fsl_flexcan.h> CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see _flexcan_fd_frame_length.

Public Members

uint32_t idhit

Note: ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using FLEXCAN_FixEnhancedRxFifoFrameIdHit API is recommended to ensure this idhit value is correct. CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).

struct _flexcan_timing_config

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

uint32_t preDivider

Classic CAN or CAN FD nominal phase bit rate prescaler.

uint32_t rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint32_t phaseSeg1

Classic CAN or CAN FD nominal phase Segment 1.

uint32_t phaseSeg2

Classic CAN or CAN FD nominal phase Segment 2.

uint32_t propSeg

Classic CAN or CAN FD nominal phase Propagation Segment.

uint32_t fpreDivider

CAN FD data phase bit rate prescaler.

uint32_t frJumpwidth

CAN FD data phase Re-sync Jump Width.

uint32_t fphaseSeg1

CAN FD data phase Phase Segment 1.

uint32_t fphaseSeg2

CAN FD data phase Phase Segment 2.

uint32_t fpropSeg

CAN FD data phase Propagation Segment.

`struct _flexcan_config`
#include <fsl_flexcan.h> FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

Public Members

flexcan_clock_source_t clkSrc

Clock source for FlexCAN Protocol Engine.

flexcan_wake_up_source_t wakeupSrc

Wake up source selection.

`uint8_t` maxMbNum

The maximum number of Message Buffers used by user.

`bool` enableLoopBack

Enable or Disable Loop Back Self Test Mode.

`bool` enableTimerSync

Enable or Disable Timer Synchronization.

`bool` enableSelfWakeup

Enable or Disable Self Wakeup Mode.

`bool` enableIndividMask

Enable or Disable Rx Individual Mask and Queue feature.

`bool` disableSelfReception

Enable or Disable Self Reflection.

`bool` enableListenOnlyMode

Enable or Disable Listen Only Mode.

`bool` enableDoze

Enable or Disable Doze Mode.

`bool` enablePretendedeNetworking

Enable or Disable the Pretended Networking mode.

`bool` enableMemoryErrorControl

Enable or Disable the memory errors detection and correction mechanism.

`bool` enableNonCorrectableErrorEnterFreeze

Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.

`bool` enableTransceiverDelayMeasure

Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.

`bool` enableRemoteRequestFrameStored

true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response Frame.

struct `_flexcan_rx_mb_config`

#include <fsl_flexcan.h> FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

Public Members

uint32_t id

CAN Message Buffer Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

flexcan_frame_format_t format

CAN Frame Identifier format(Standard of Extend).

flexcan_frame_type_t type

CAN Frame Type(Data or Remote for classical CAN only).

struct `_flexcan_pn_config`

#include <fsl_flexcan.h> FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

Public Members

bool enableTimeout

Enable or Disable timeout event trigger wakeup.

uint16_t timeoutValue

The timeout value that generates a wakeup event, the counter timer is incremented based on 64 times the CAN Bit Time unit.

bool enableMatch

Enable or Disable match event trigger wakeup.

flexcan_pn_match_source_t matchSrc

Selects the match source (ID and/or data match) to trigger wakeup.

uint8_t matchNum

The number of times a given message must match the predefined ID and/or data before generating a wakeup event, range in 0x1 ~ 0xFF.

flexcan_pn_match_mode_t idMatchMode

The ID match type.

flexcan_pn_match_mode_t dataMatchMode

The data match type.

uint32_t idLower

The ID target values 1 which used either for ID match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in ID match “range detection”.

uint32_t idUpper

The ID target values 2 which used only as the upper limit value in ID match “range detection” or used to store the ID mask in “equal to”.

uint8_t lengthLower

The lower limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

uint8_t lengthUpper

The upper limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

struct _flexcan_rx_fifo_config

#include <fsl_flexcan.h> FlexCAN Legacy Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

uint8_t idFilterNum

The FlexCAN Legacy Rx FIFO Filter elements quantity.

flexcan_rx_fifo_filter_type_t idFilterType

The FlexCAN Legacy Rx FIFO Filter type.

flexcan_rx_fifo_priority_t priority

The FlexCAN Legacy Rx FIFO receive priority.

struct _flexcan_enhanced_rx_fifo_std_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see *_flexcan_fd_frame_length*, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use *DLC_LENGTH_DECODE(length)* macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct _flexcan_enhanced_rx_fifo_ext_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length ≤ 8 , it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct `_flexcan_enhanced_rx_fifo_config`

`#include <fsl_flexcan.h>` FlexCAN Enhanced Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to `idFilterNum`. There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupie 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

uint8_t idFilterPairNum

`idFilterPairNum` is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

uint8_t extendIdFilterNum

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, `extendIdFilterNum` need less than or equal to `idFilterPairNum`.

uint8_t fifoWatermark

$(\text{fifoWatermark} + 1)$ is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

flexcan_efifo_dma_per_read_length_t dmaPerReadLength

Define the length of each read of the Enhanced RX FIFO element by the DAM, see `_flexcan_fd_frame_length`.

flexcan_rx_fifo_priority_t priority

The FlexCAN Enhanced Rx FIFO receive priority.

struct `_flexcan_mb_transfer`

`#include <fsl_flexcan.h>` FlexCAN Message Buffer transfer.

Public Members

flexcan_frame_t *frame

The buffer of CAN Message to be transfer.

uint8_t mbIdx

The index of Message buffer used to transfer Message.

```
struct _flexcan_fifo_transfer
#include <fsl_flexcan.h> FlexCAN Rx FIFO transfer.
```

Public Members

flexcan_fd_frame_t *framefd
The buffer of CAN Message to be received from Enhanced Rx FIFO.

flexcan_frame_t *frame
The buffer of CAN Message to be received from Legacy Rx FIFO.

size_t frameNum
Number of CAN Message need to be received from Legacy or Enhanced Rx FIFO.

```
struct _flexcan_handle
#include <fsl_flexcan.h> FlexCAN handle structure.
```

Public Members

flexcan_transfer_callback_t callback
Callback function.

void *userData
FlexCAN callback function parameter.

flexcan_frame_t *volatile mbFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN data from Message Buffers.

flexcan_fd_frame_t *volatile mbFDFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN FD data from Message Buffers.

flexcan_frame_t *volatile rxFifoFrameBuf
The buffer for received CAN data from Legacy Rx FIFO.

flexcan_fd_frame_t *volatile rxFifoFDFrameBuf
The buffer for received CAN FD data from Enhanced Rx FIFO.

size_t rxFifoFrameNum
The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.

size_t rxFifoTransferTotalNum
Total CAN Message number need to be received from Legacy or Enhanced Rx FIFO.

volatile uint8_t mbState[CAN_WORD1_COUNT]
Message Buffer transfer state.

volatile uint8_t rxFifoState
Rx FIFO transfer state.

volatile uint32_t timestamp[CAN_WORD1_COUNT]
Mailbox transfer timestamp.

```
struct byteStatus
```


Public Members

bool byteIsRead

The byte n (0~3) was read or not. The type of error and which bit in byte (n) is affected by the error.

struct ____unnamed18____

Public Members

uint32_t timestamp

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length

CAN frame data length in bytes (Range: 0~8).

uint32_t type

CAN Frame Type(DATA or REMOTE).

uint32_t format

CAN Frame Identifier(STD or EXT format).

uint32_t ____pad0____

Reserved.

uint32_t idhit

CAN Rx FIFO filter hit id(This value is only used in Rx FIFO receive mode).

struct ____unnamed20____

Public Members

uint32_t id

CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

uint32_t ____pad0____

Reserved.

union ____unnamed22____

Public Members

struct __flexcan_frame

struct __flexcan_frame

struct ____unnamed24____

Public Members

uint32_t dataWord0

CAN Frame payload word0.

uint32_t dataWord1

CAN Frame payload word1.

struct ____unnamed26____

Public Members

uint8_t dataByte3
CAN Frame payload byte3.

uint8_t dataByte2
CAN Frame payload byte2.

uint8_t dataByte1
CAN Frame payload byte1.

uint8_t dataByte0
CAN Frame payload byte0.

uint8_t dataByte7
CAN Frame payload byte7.

uint8_t dataByte6
CAN Frame payload byte6.

uint8_t dataByte5
CAN Frame payload byte5.

uint8_t dataByte4
CAN Frame payload byte4.

struct ____unnamed28____

Public Members

uint32_t timestamp
FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length
CAN FD frame data length code (DLC), range see `_flexcan_fd_frame_length`, When the length ≤ 8 , it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

uint32_t type
CAN Frame Type(DATA only).

uint32_t format
CAN Frame Identifier(STD or EXT format).

uint32_t srr
Substitute Remote request.

uint32_t esi
Error State Indicator.

uint32_t brs
Bit Rate Switch.

uint32_t edl
Extended Data Length.

struct ____unnamed30____

Public Members

uint32_t id

CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

uint32_t __pad0__

Reserved.

union __unnamed32__

Public Members

struct __flexcan_fd_frame

struct __flexcan_fd_frame

struct __unnamed34__

Public Members

uint32_t dataWord[16]

CAN FD Frame payload, 16 double word maximum.

struct __unnamed36__

Public Members

uint8_t dataByte3

CAN Frame payload byte3.

uint8_t dataByte2

CAN Frame payload byte2.

uint8_t dataByte1

CAN Frame payload byte1.

uint8_t dataByte0

CAN Frame payload byte0.

uint8_t dataByte7

CAN Frame payload byte7.

uint8_t dataByte6

CAN Frame payload byte6.

uint8_t dataByte5

CAN Frame payload byte5.

uint8_t dataByte4

CAN Frame payload byte4.

union __unnamed38__

Public Members

struct __flexcan_config

struct __flexcan_config

struct ____unnamed40____

Public Members

uint32_t baudRate

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t baudRateFD

FlexCAN FD bit rate in bps, for CANFD data phase.

struct ____unnamed42____

Public Members

uint32_t bitRate

FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t bitRateFD

FlexCAN FD bit rate in bps, for CANFD data phase.

union ____unnamed44____

Public Members

struct __flexcan_pn_config

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

struct __flexcan_pn_config

struct ____unnamed48____

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range detection”.

Public Members

uint32_t lowerWord0

CAN Frame payload word0.

uint32_t lowerWord1

CAN Frame payload word1.

struct ____unnamed50____

Public Members

uint8_t lowerByte3
CAN Frame payload byte3.

uint8_t lowerByte2
CAN Frame payload byte2.

uint8_t lowerByte1
CAN Frame payload byte1.

uint8_t lowerByte0
CAN Frame payload byte0.

uint8_t lowerByte7
CAN Frame payload byte7.

uint8_t lowerByte6
CAN Frame payload byte6.

uint8_t lowerByte5
CAN Frame payload byte5.

uint8_t lowerByte4
CAN Frame payload byte4.

union __unnamed46__

Public Members

struct __flexcan_pn_config

< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

struct __flexcan_pn_config

struct __unnamed52__

< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

Public Members

uint32_t upperWord0
CAN Frame payload word0.

uint32_t upperWord1
CAN Frame payload word1.

struct __unnamed54__

Public Members

uint8_t upperByte3
CAN Frame payload byte3.

uint8_t upperByte2
CAN Frame payload byte2.

uint8_t upperByte1
CAN Frame payload byte1.

uint8_t upperByte0
CAN Frame payload byte0.

uint8_t upperByte7
CAN Frame payload byte7.

uint8_t upperByte6
CAN Frame payload byte6.

uint8_t upperByte5
CAN Frame payload byte5.

uint8_t upperByte4
CAN Frame payload byte4.

2.17 FlexCAN eDMA Driver

void FLEXCAN__TransferCreateHandleEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_edma_transfer_callback_t callback, void *userData, edma_handle_t *rxFifoEdmaHandle)

Initializes the FlexCAN handle, which is used in transactional functions.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

void FLEXCAN__PrepareTransfConfiguration(CAN_Type *base, flexcan_fifo_transfer_t *pFifoXfer, edma_transfer_config_t *pEdmaConfig)

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

status_t FLEXCAN__StartTransferDatafromRxFIFO(CAN_Type *base, flexcan_edma_handle_t *handle, edma_transfer_config_t *pEdmaConfig)

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer configuration and Start eDMA transfer

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

Return values

- kStatus__Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN__TransferReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus__Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN__TransferGetReceiveFifoCountEMDA(CAN_Type *base, flexcan_edma_handle_t *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus__InvalidArgument – count is Invalid.
- kStatus__Success – Successfully return the count.

void FLEXCAN__TransferAbortReceiveFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle)

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.

status_t FLEXCAN__TransferReceiveEnhancedFifoEDMA(CAN_Type *base, flexcan_edma_handle_t *handle, flexcan_fifo_transfer_t *pFifoXfer)

Receives the CAN FD Message from the Enhanced Rx FIFO using eDMA.

This function receives the CAN FD Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

```
static inline status_t FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA(CAN_Type *base,  
                                                                    flex-  
                                                                    can_edma_handle_t  
                                                                    *handle, size_t  
                                                                    *count)
```

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

FSL_FLEXCAN_EDMA_DRIVER_VERSION

FlexCAN EDMA driver version.

```
typedef struct flexcan_edma_handle flexcan_edma_handle_t
```

```
typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, flexcan_edma_handle_t  
*handle, status_t status, void *userData)
```

FlexCAN transfer callback function.

```
struct flexcan_edma_handle
```

```
#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.
```

Public Members

flexcan_edma_transfer_callback_t callback
Callback function.

void *userData

FlexCAN callback function parameter.

edma_handle_t *rxFifoEdmaHandle

The EDMA handler for Rx FIFO.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

size_t frameNum

The number of messages that need to be received.

flexcan_fd_frame_t *framefd

Point to the buffer of CAN Message to be received from Enhanced Rx FIFO.

2.18 FlexIO: FlexIO Driver

2.19 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

- userConfig – pointer to flexio_config_t structure

void FLEXIO_Init(FLEXIO_Type *base, const *flexio_config_t* *userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_GetDefaultConfig().

Example

```
flexio_config_t config = {
    .enableFlexio = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio_config_t structure

void FLEXIO_Deinit(FLEXIO_Type *base)

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

Note: After calling this API, call the FLEXIO_Init to use the FlexIO module.

Parameters

- base – FlexIO peripheral base address

uint32_t FLEXIO_GetInstance(FLEXIO_Type *base)

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

void FLEXIO_Reset(FLEXIO_Type *base)

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

static inline void FLEXIO_Enable(FLEXIO_Type *base, bool enable)

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns

FlexIO pin input data

static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns

current State pointer

void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const *flexio_shifter_config_t* *shifterConfig)

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {  
    .timerSelect = 0,  
    .timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,  
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,  
    .pinPolarity = kFLEXIO_PinActiveLow,  
    .shifterMode = kFLEXIO_ShifterModeTransmit,  
    .inputSource = kFLEXIO_ShifterInputFromPin,  
    .shifterStop = kFLEXIO_ShifterStopBitHigh,  
    .shifterStart = kFLEXIO_ShifterStartBitLow  
};  
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address

- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

```
void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t
                          *timerConfig)
```

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
    .triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFThSTAT(0),
    .triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
    .triggerSource = kFLEXIO_TimerTriggerSourceInternal,
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
    .pinSelect = 0,
    .pinPolarity = kFLEXIO_PinActiveHigh,
    .timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
    .timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
    .timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
    .timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
    .timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
    .timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
    .timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
    .timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

```
static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index,
                                       flexio_timer_decrement_source_t clocksource)
```

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

```
static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)
Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)

Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)

Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

```
static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)
```

Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

```
static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

```
static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)
```

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$
- enable – True to enable, false to disable.

```
uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, flexio_shifter_buffer_type_t type, uint8_t index)
```

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio_shifter_buffer_type_t
- index – Shifter index

Returns

Corresponding shifter buffer index

```
status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, flexio_isr_t isr)
```

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`status_t` FLEXIO_UnregisterHandleIRQ(void *base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- `base` – Pointer to the FlexIO simulated peripheral type.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 0.

Parameters

- `base` – FlexIO peripheral base address
- `mask` – FLEXIO pin number mask

static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 1.

Parameters

- `base` – FlexIO peripheral base address
- `mask` – FLEXIO pin number mask

static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- `base` – FlexIO peripheral base address
- `mask` – FLEXIO pin number mask

static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- `base` – FlexIO peripheral base address
- `pin` – FLEXIO pin number.
- `output` – FLEXIO pin output logic level.
 - 0: corresponding pin output low-logic level.
 - 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- `base` – FlexIO peripheral base address
- `pin` – FLEXIO pin number.

```
static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)
```

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

```
static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)
```

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

```
static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)
```

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

```
static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)
```

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

```
static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)
```

Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)

Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL_FLEXIO_DRIVER_VERSION

FlexIO driver version.

enum __flexio_timer_trigger_polarity

Define time of timer trigger polarity.

Values:

enumerator kFLEXIO_TimerTriggerPolarityActiveHigh
Active high.

enumerator kFLEXIO_TimerTriggerPolarityActiveLow
Active low.

enum __flexio_timer_trigger_source

Define type of timer trigger source.

Values:

enumerator kFLEXIO_TimerTriggerSourceExternal
External trigger selected.

enumerator kFLEXIO_TimerTriggerSourceInternal
Internal trigger selected.

enum __flexio_pin_config

Define type of timer/shifter pin configuration.

Values:

enumerator kFLEXIO_PinConfigOutputDisabled
Pin output disabled.

enumerator kFLEXIO_PinConfigOpenDrainOrBidirection
Pin open drain or bidirectional output enable.

enumerator kFLEXIO_PinConfigBidirectionOutputData
Pin bidirectional output data.

enumerator kFLEXIO_PinConfigOutput
Pin output.

enum __flexio_pin_polarity

Definition of pin polarity.

Values:

enumerator kFLEXIO_PinActiveHigh
Active high.

enumerator kFLEXIO_PinActiveLow
Active low.

enum __flexio_timer_mode

Define type of timer work mode.

Values:

enumerator kFLEXIO__TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO__TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO__TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO__TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO__TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum __flexio_timer_output
Define type of timer initial output or timer reset condition.
Values:

enumerator kFLEXIO__TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO__TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO__TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO__TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum __flexio_timer_decrement_source
Define type of timer decrement.
Values:

enumerator kFLEXIO__TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO__TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO__TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO__TimerDecSrcOnTriggerInputShiftTriggerInput
Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum __flexio_timer_reset_condition
Define type of timer reset condition.
Values:

enumerator kFLEXIO__TimerResetNever
Timer never reset.

enumerator kFLEXIO__TimerResetOnTimerPinEqualToTimerOutput
Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerTriggerEqualToTimerOutput
Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerPinRisingEdge

Timer reset on Timer Pin rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerRisingEdge

Timer reset on Trigger rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerBothEdge

Timer reset on Trigger rising or falling edge.

enum _flexio_timer_disable_condition

Define type of timer disable condition.

Values:

enumerator kFLEXIO__TimerDisableNever

Timer never disabled.

enumerator kFLEXIO__TimerDisableOnPreTimerDisable

Timer disabled on Timer N-1 disable.

enumerator kFLEXIO__TimerDisableOnTimerCompare

Timer disabled on Timer compare.

enumerator kFLEXIO__TimerDisableOnTimerCompareTriggerLow

Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO__TimerDisableOnPinBothEdge

Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO__TimerDisableOnPinBothEdgeTriggerHigh

Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO__TimerDisableOnTriggerFallingEdge

Timer disabled on Trigger falling edge.

enum _flexio_timer_enable_condition

Define type of timer enable condition.

Values:

enumerator kFLEXIO__TimerEnabledAlways

Timer always enabled.

enumerator kFLEXIO__TimerEnableOnPrevTimerEnable

Timer enabled on Timer N-1 enable.

enumerator kFLEXIO__TimerEnableOnTriggerHigh

Timer enabled on Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerHighPinHigh

Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO__TimerEnableOnPinRisingEdge

Timer enabled on Pin rising edge.

enumerator kFLEXIO__TimerEnableOnPinRisingEdgeTriggerHigh

Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerRisingEdge

Timer enabled on Trigger rising edge.

enumerator kFLEXIO__TimerEnableOnTriggerBothEdge

Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition

Define type of timer stop bit generate condition.

Values:

enumerator kFLEXIO_TimerStopBitDisabled

Stop bit disabled.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompare

Stop bit is enabled on timer compare.

enumerator kFLEXIO_TimerStopBitEnableOnTimerDisable

Stop bit is enabled on timer disable.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompareDisable

Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition

Define type of timer start bit generate condition.

Values:

enumerator kFLEXIO_TimerStartBitDisabled

Start bit disabled.

enumerator kFLEXIO_TimerStartBitEnabled

Start bit enabled.

enum _flexio_timer_output_state

FlexIO as PWM channel output state.

Values:

enumerator kFLEXIO_PwmLow

The output state of PWM channel is low

enumerator kFLEXIO_PwmHigh

The output state of PWM channel is high

enum _flexio_shifter_timer_polarity

Define type of timer polarity for shifter control.

Values:

enumerator kFLEXIO_ShifterTimerPolarityOnPositive

Shift on positive edge of shift clock.

enumerator kFLEXIO_ShifterTimerPolarityOnNegative

Shift on negative edge of shift clock.

enum _flexio_shifter_mode

Define type of shifter working mode.

Values:

enumerator kFLEXIO_ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO_ShifterModeReceive

Receive mode.

enumerator kFLEXIO_ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO__ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO__ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO__ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO__ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio_shifter_input_source

Define type of shifter input source.

Values:

enumerator kFLEXIO__ShifterInputFromPin

Shifter input from pin.

enumerator kFLEXIO__ShifterInputFromNextShifterOutput

Shifter input from Shifter N+1.

enum _flexio_shifter_stop_bit

Define of STOP bit configuration.

Values:

enumerator kFLEXIO__ShifterStopBitDisable

Disable shifter stop bit.

enumerator kFLEXIO__ShifterStopBitLow

Set shifter stop bit to logic low level.

enumerator kFLEXIO__ShifterStopBitHigh

Set shifter stop bit to logic high level.

enum _flexio_shifter_start_bit

Define type of START bit configuration.

Values:

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnEnable

Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnShift

Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO__ShifterStartBitLow

Set shifter start bit to logic low level.

enumerator kFLEXIO__ShifterStartBitHigh

Set shifter start bit to logic high level.

enum _flexio_shifter_buffer_type

Define FlexIO shifter buffer type.

Values:

enumerator kFLEXIO__ShifterBuffer

Shifter Buffer N Register.

enumerator kFLEXIO__ShifterBufferBitSwapped

Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferByteSwapped
Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferBitByteSwapped
Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO__ShifterBufferNibbleByteSwapped
Shifter Buffer N Nibble Byte Swapped Register.

enumerator kFLEXIO__ShifterBufferHalfWordSwapped
Shifter Buffer N Half Word Swapped Register.

enumerator kFLEXIO__ShifterBufferNibbleSwapped
Shifter Buffer N Nibble Swapped Register.

enum _flexio_gpio_direction
FLEXIO gpio direction definition.

Values:

enumerator kFLEXIO__DigitalInput
Set current pin as digital input

enumerator kFLEXIO__DigitalOutput
Set current pin as digital output

enum _flexio_pin_input_config
FLEXIO gpio input config.

Values:

enumerator kFLEXIO__InputInterruptDisabled
Interrupt request is disabled.

enumerator kFLEXIO__InputInterruptEnable
Interrupt request is enable.

enumerator kFLEXIO__FlagRisingEdgeEnable
Input pin flag on rising edge.

enumerator kFLEXIO__FlagFallingEdgeEnable
Input pin flag on falling edge.

typedef enum _flexio_timer_trigger_polarity flexio_timer_trigger_polarity_t
Define time of timer trigger polarity.

typedef enum _flexio_timer_trigger_source flexio_timer_trigger_source_t
Define type of timer trigger source.

typedef enum _flexio_pin_config flexio_pin_config_t
Define type of timer/shifter pin configuration.

typedef enum _flexio_pin_polarity flexio_pin_polarity_t
Definition of pin polarity.

typedef enum _flexio_timer_mode flexio_timer_mode_t
Define type of timer work mode.

typedef enum _flexio_timer_output flexio_timer_output_t
Define type of timer initial output or timer reset condition.

typedef enum _flexio_timer_decrement_source flexio_timer_decrement_source_t
Define type of timer decrement.

typedef enum *_flexio_timer_reset_condition* flexio_timer_reset_condition_t

Define type of timer reset condition.

typedef enum *_flexio_timer_disable_condition* flexio_timer_disable_condition_t

Define type of timer disable condition.

typedef enum *_flexio_timer_enable_condition* flexio_timer_enable_condition_t

Define type of timer enable condition.

typedef enum *_flexio_timer_stop_bit_condition* flexio_timer_stop_bit_condition_t

Define type of timer stop bit generate condition.

typedef enum *_flexio_timer_start_bit_condition* flexio_timer_start_bit_condition_t

Define type of timer start bit generate condition.

typedef enum *_flexio_timer_output_state* flexio_timer_output_state_t

FlexIO as PWM channel output state.

typedef enum *_flexio_shifter_timer_polarity* flexio_shifter_timer_polarity_t

Define type of timer polarity for shifter control.

typedef enum *_flexio_shifter_mode* flexio_shifter_mode_t

Define type of shifter working mode.

typedef enum *_flexio_shifter_input_source* flexio_shifter_input_source_t

Define type of shifter input source.

typedef enum *_flexio_shifter_stop_bit* flexio_shifter_stop_bit_t

Define of STOP bit configuration.

typedef enum *_flexio_shifter_start_bit* flexio_shifter_start_bit_t

Define type of START bit configuration.

typedef enum *_flexio_shifter_buffer_type* flexio_shifter_buffer_type_t

Define FlexIO shifter buffer type.

typedef struct *_flexio_config* flexio_config_t

Define FlexIO user configuration structure.

typedef struct *_flexio_timer_config* flexio_timer_config_t

Define FlexIO timer configuration structure.

typedef struct *_flexio_shifter_config* flexio_shifter_config_t

Define FlexIO shifter configuration structure.

typedef enum *_flexio_gpio_direction* flexio_gpio_direction_t

FLEXIO gpio direction definition.

typedef enum *_flexio_pin_input_config* flexio_pin_input_config_t

FLEXIO gpio input config.

typedef struct *_flexio_gpio_config* flexio_gpio_config_t

The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

typedef void (*flexio_isr_t)(void *base, void *handle)

typedef for FlexIO simulated driver interrupt handler.

FLEXIO_Type *const s_flexioBases[]

Pointers to flexio bases for each instance.

```
const clock_ip_name_t s_flexioClocks[]
```

Pointers to flexio clocks for each instance.

```
void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, flexio_gpio_config_t *config)
```

Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the FLEXIO_SetPinConfig() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalInput,
    0U,
    kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalOutput,
    0U,
    0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

```
FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)
```

Calculate FlexIO timer trigger.

```
FLEXIO_TIMER_TRIGGER_SEL_SHIFtnSTAT(x)
```

```
FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)
```

```
struct _flexio_config_
```

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members

```
bool enableFlexio
```

Enable/disable FlexIO module

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
struct _flexio_timer_config
```

#include <fsl_flexio.h> Define FlexIO timer configuration structure.

Public Members

`uint32_t triggerSelect`

The internal trigger selection number using MACROs.

`flexio_timer_trigger_polarity_t triggerPolarity`

Trigger Polarity.

`flexio_timer_trigger_source_t triggerSource`

Trigger Source, internal (see 'trgsel') or external.

`flexio_pin_config_t pinConfig`

Timer Pin Configuration.

`uint32_t pinSelect`

Timer Pin number Select.

`flexio_pin_polarity_t pinPolarity`

Timer Pin Polarity.

`flexio_timer_mode_t timerMode`

Timer work Mode.

`flexio_timer_output_t timerOutput`

Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

`flexio_timer_decrement_source_t timerDecrement`

Configures the source of the Timer decrement and the source of the Shift clock.

`flexio_timer_reset_condition_t timerReset`

Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

`flexio_timer_disable_condition_t timerDisable`

Configures the condition that causes the Timer to be disabled and stop decrementing.

`flexio_timer_enable_condition_t timerEnable`

Configures the condition that causes the Timer to be enabled and start decrementing.

`flexio_timer_stop_bit_condition_t timerStop`

Timer STOP Bit generation.

`flexio_timer_start_bit_condition_t timerStart`

Timer STRAT Bit generation.

`uint32_t timerCompare`

Value for Timer Compare N Register.

`struct _flexio_shifter_config`

`#include <fsl_flexio.h>` Define FlexIO shifter configuration structure.

Public Members

`uint32_t timerSelect`

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

`flexio_shifter_timer_polarity_t timerPolarity`

Timer Polarity.

flexio_pin_config_t pinConfig
Shifter Pin Configuration.

uint32_t pinSelect
Shifter Pin number Select.

flexio_pin_polarity_t pinPolarity
Shifter Pin Polarity.

flexio_shifter_mode_t shifterMode
Configures the mode of the Shifter.

uint32_t parallelWidth
Configures the parallel width when using parallel mode.

flexio_shifter_input_source_t inputSource
Selects the input source for the shifter.

flexio_shifter_stop_bit_t shifterStop
Shifter STOP bit.

flexio_shifter_start_bit_t shifterStart
Shifter START bit.

struct __flexio_gpio_config

#include <fsl_flexio.h> The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

flexio_gpio_direction_t pinDirection
FLEXIO pin direction, input or output

uint8_t outputLogic
Set a default output logic, which has no use in input

uint8_t inputConfig
Set an input config

2.20 FlexIO eDMA I2S Driver

```
void FLEXIO_I2S_TransferTxCreateHandleEDMA(FLEXIO_I2S_Type *base,  
                                           flexio_i2s_edma_handle_t *handle,  
                                           flexio_i2s_edma_callback_t callback, void  
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.

- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferRxCreateHandleEDMA(FLEXIO_I2S_Type *base,  
                                           flexio_i2s_edma_handle_t *handle,  
                                           flexio_i2s_edma_callback_t callback, void  
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferSetFormatEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t  
                                       *handle, flexio_i2s_format_t *format, uint32_t  
                                       srcClock_Hz)
```

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

```
status_t FLEXIO_I2S_TransferSendEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t  
                                       *handle, flexio_i2s_transfer_t *xfer)
```

Performs a non-blocking FlexIO I2S transfer using DMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- `kStatus__Success` – Start a FlexIO I2S eDMA send successfully.
- `kStatus__InvalidArgument` – The input arguments is invalid.
- `kStatus__TxBusy` – FlexIO I2S is busy sending data.

`status_t` FLEXIO_I2S_TransferReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S receive using eDMA.

Note: This interface returned immediately after transfer initiates. Users should call `FLEXIO_I2S_GetReceiveRemainingBytes` to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- `kStatus__Success` – Start a FlexIO I2S eDMA receive successfully.
- `kStatus__InvalidArgument` – The input arguments is invalid.
- `kStatus__RxBusy` – FlexIO I2S is busy receiving data.

`void` FLEXIO_I2S_TransferAbortSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S transfer using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

`void` FLEXIO_I2S_TransferAbortReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S receive using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

`status_t` FLEXIO_I2S_TransferGetSendCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be sent.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

Return values

- `kStatus__Success` – Succeed get the transfer count.

- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

```
status_t FLEXIO_I2S_TransferGetReceiveCountEDMA(FLEXIO_I2S_Type *base,
                                                flexio_i2s_edma_handle_t *handle, size_t
                                                *count)
```

Get the remaining bytes to be received.

Parameters

- `base` – FlexIO I2S peripheral base address.
- `handle` – FlexIO I2S DMA handle pointer.
- `count` – Bytes received.

Return values

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

FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION

FlexIO I2S EDMA driver version 2.1.9.

```
typedef struct _flexio_i2s_edma_handle flexio_i2s_edma_handle_t
```

```
typedef void (*flexio_i2s_edma_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
*handle, status_t status, void *userData)
```

FlexIO I2S eDMA transfer callback function for finish and error.

```
struct _flexio_i2s_edma_handle
```

`#include <fsl_flexio_i2s_edma.h>` FlexIO I2S DMA transfer handle, users should not touch the content of the handle.

Public Members

```
edma_handle_t *dmaHandle
```

DMA handler for FlexIO I2S send

```
uint8_t bytesPerFrame
```

Bytes in a frame

```
uint8_t nbytes
```

eDMA minor byte transfer count initially configured.

```
uint32_t state
```

Internal state for FlexIO I2S eDMA transfer

```
flexio_i2s_edma_callback_t callback
```

Callback for users while transfer finish or error occurred

```
void *userData
```

User callback parameter

```
edma_tcd_t tcd[(4U) + 1U]
```

TCD pool for eDMA transfer.

```
flexio_i2s_transfer_t queue[(4U)]
```

Transfer queue storing queued transfer.

```
size_t transferSize[(4U)]
```

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.21 FlexIO eDMA SPI Driver

```
status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                    flexio_spi_master_edma_handle_t  
                                                    *handle,  
                                                    flexio_spi_master_edma_transfer_callback_t  
                                                    callback, void *userData,  
                                                    edma_handle_t *txHandle,  
                                                    edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI master eDMA handle.

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

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

```
status_t FLEXIO_SPI_MasterTransferEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle,  
                                         flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – FlexIO SPI is not idle, is running another transfer.

```
void FLEXIO_SPI_MasterTransferAbortEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – FlexIO SPI eDMA handle pointer.

```
status_t FLEXIO_SPI_MasterTransferGetCountEDMA(FLEXIO_SPI_Type *base,  
                                                flexio_spi_master_edma_handle_t *handle,  
                                                size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – FlexIO SPI eDMA handle pointer.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

```
static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                           flexio_spi_slave_edma_handle_t  
                                                           *handle,  
                                                           flexio_spi_slave_edma_transfer_callback_t  
                                                           callback, void *userData,  
                                                           edma_handle_t *txHandle,  
                                                           edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- `base` – Pointer to `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to `flexio_spi_slave_edma_handle_t` structure to store the transfer state.
- `callback` – SPI callback, NULL means no callback.
- `userData` – callback function parameter.
- `txHandle` – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- `rxHandle` – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

```
status_t FLEXIO_SPI_SlaveTransferEDMA(FLEXIO_SPI_Type *base,  
                                       flexio_spi_slave_edma_handle_t *handle,  
                                       flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_SlaveGetTransferCountEDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(FLEXIO_SPI_Type *base,  
                                                    flexio_spi_slave_edma_handle_t  
                                                    *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCountEDMA(FLEXIO_SPI_Type *base,  
                                                           flexio_spi_slave_edma_handle_t  
                                                           *handle, size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION

FlexIO SPI EDMA driver version.

```
typedef struct flexio_spi_master_edma_handle flexio_spi_master_edma_handle_t  
typedef for flexio_spi_master_edma_handle_t in advance.
```

```
typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t  
Slave handle is the same with master handle.
```

```
typedef void (*flexio_spi_master_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,  
                                                           flexio_spi_master_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,  
                                                           flexio_spi_slave_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
struct flexio_spi_master_edma_handle
```

```
#include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch  
the content of the handle.
```

Public Members

`size_t` transferSize
Total bytes to be transferred.

`uint8_t` nbytes
eDMA minor byte transfer count initially configured.

`bool` txInProgress
Send transfer in progress

`bool` rxInProgress
Receive transfer in progress

`edma_handle_t` *txHandle
DMA handler for SPI send

`edma_handle_t` *rxHandle
DMA handler for SPI receive

`flexio_spi_master_edma_transfer_callback_t` callback
Callback for SPI DMA transfer

`void` *userData
User Data for SPI DMA callback

2.22 FlexIO eDMA UART Driver

`status_t` FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base,
 flexio_uart_edma_handle_t *handle,
 flexio_uart_edma_transfer_callback_t
 callback, void *userData, *edma_handle_t*
 *txEdmaHandle, *edma_handle_t*
 *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to *FLEXIO_UART_Type*.
- handle – Pointer to *flexio_uart_edma_handle_t* structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO SPI eDMA type/handle table out of range.

`status_t` FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base,
 flexio_uart_edma_handle_t *handle,
 flexio_uart_transfer_t *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

```
status_t FLEXIO_UART_TransferReceiveEDMA(FLEXIO_UART_Type *base,  
                                          flexio_uart_edma_handle_t *handle,  
                                          flexio_uart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

```
void FLEXIO_UART_TransferAbortSendEDMA(FLEXIO_UART_Type *base,  
                                       flexio_uart_edma_handle_t *handle)
```

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
void FLEXIO_UART_TransferAbortReceiveEDMA(FLEXIO_UART_Type *base,  
                                           flexio_uart_edma_handle_t *handle)
```

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
status_t FLEXIO_UART_TransferGetSendCountEDMA(FLEXIO_UART_Type *base,  
                                              flexio_uart_edma_handle_t *handle,  
                                              size_t *count)
```

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

`status_t FLEXIO_UART_TransferGetReceiveCountEDMA(FLEXIO_UART_Type *base, flexio_uart_edma_handle_t *handle, size_t *count)`

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- `base` – Pointer to `FLEXIO_UART_Type`
- `handle` – Pointer to `flexio_uart_edma_handle_t` structure
- `count` – Number of bytes received so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

`FSL_FLEXIO_UART_EDMA_DRIVER_VERSION`

FlexIO UART EDMA driver version.

`typedef struct flexio_uart_edma_handle flexio_uart_edma_handle_t`

`typedef void (*flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_edma_handle_t *handle, status_t status, void *userData)`

UART transfer callback function.

`struct flexio_uart_edma_handle`

`#include <fsl_flexio_uart_edma.h>` UART eDMA handle.

Public Members

`flexio_uart_edma_transfer_callback_t` callback

Callback function.

`void *userData`

UART callback function parameter.

`size_t txDataSizeAll`

Total bytes to be sent.

`size_t rxDataSizeAll`

Total bytes to be received.

`edma_handle_t *txEdmaHandle`

The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`

The eDMA RX channel used.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.23 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {  
.flexioBase = FLEXIO,  
.SDAPinIndex = 0,  
.SCLPinIndex = 1,  
.shifterIndex = {0,1},  
.timerIndex = {0,1}  
};  
flexio_i2c_master_config_t config = {  
.enableInDoze = false,  
.enableInDebug = true,  
.enableFastAccess = false,  
.baudRate_Bps = 100000  
};  
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND `_flexio_i2c_master_status_flags` can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_I2C_RxFullFlag
 - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
 - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

```
void FLEXIO_I2C_MasterSetBaudRate(FLEXIO_I2C_Type *base, uint32_t baudRate_Bps,  
                                uint32_t srcClock_Hz)
```

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

```
void FLEXIO_I2C_MasterStart(FLEXIO_I2C_Type *base, uint8_t address, flexio_i2c_direction_t  
                           direction)
```

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the *kFLEXIO_I2C_RxFullFlag* status is asserted before calling this API.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in *flexio_i2c_direction_t*:
 - *kFLEXIO_I2C_Write*: Transmit
 - *kFLEXIO_I2C_Read*: Receive

```
void FLEXIO_I2C_MasterStop(FLEXIO_I2C_Type *base)
```

Sends the stop signal on the bus.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

```
void FLEXIO_I2C_MasterRepeatedStart(FLEXIO_I2C_Type *base)
```

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

```
void FLEXIO_I2C_MasterAbortStop(FLEXIO_I2C_Type *base)
```

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.

```
void FLEXIO_I2C_MasterEnableAck(FLEXIO_I2C_Type *base, bool enable)
```

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to *FLEXIO_I2C_Type* structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

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

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_transfer_t *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_handle_t *handle,
flexio_i2c_master_transfer_callback_t
callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.


```
status_t FLEXIO_I2C_MasterTransferNonBlocking(FLEXIO_I2C_Type *base,  
                                              flexio_i2c_master_handle_t *handle,  
                                              flexio_i2c_master_transfer_t *xfer)
```

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

```
status_t FLEXIO_I2C_MasterTransferGetCount(FLEXIO_I2C_Type *base,  
                                           flexio_i2c_master_handle_t *handle, size_t  
                                           *count)
```

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus_Success – Successfully return the count.

```
void FLEXIO_I2C_MasterTransferAbort(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t  
                                   *handle)
```

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak

NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout

Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable

Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable

Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag

Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag

Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag

Receive NAK flag.

enum _flexio_i2c_direction

Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write

Master send to slave.

enumerator kFLEXIO_I2C_Read

Master receive from slave.

typedef enum _flexio_i2c_direction flexio_i2c_direction_t

Direction of master transfer.

```
typedef struct _flexio_i2c_type FLEXIO_I2C_Type
    Define FlexIO I2C master access structure typedef.

typedef struct _flexio_i2c_master_config flexio_i2c_master_config_t
    Define FlexIO I2C master user configuration structure.

typedef struct _flexio_i2c_master_transfer flexio_i2c_master_transfer_t
    Define FlexIO I2C master transfer structure.

typedef struct _flexio_i2c_master_handle flexio_i2c_master_handle_t
    FlexIO I2C master handle typedef.

typedef void (*flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base,
flexio_i2c_master_handle_t *handle, status_t status, void *userData)
    FlexIO I2C master transfer callback typedef.

I2C_RETRY_TIMES
    Retry times for waiting flag.

struct _flexio_i2c_type
    #include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.
```

Public Members

```
FLEXIO_Type *flexioBase
    FlexIO base pointer.

uint8_t SDAPinIndex
    Pin select for I2C SDA.

uint8_t SCLPinIndex
    Pin select for I2C SCL.

uint8_t shifterIndex[2]
    Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]
    Timer index used in FlexIO I2C.

uint32_t baudrate
    Master transfer baudrate, used to calculate delay time.

struct _flexio_i2c_master_config
    #include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.
```

Public Members

```
bool enableMaster
    Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze
    Enable/disable FlexIO operation in doze mode.

bool enableInDebug
    Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
    Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to
    be at least twice the frequency of the bus clock.
```

uint32_t baudRate_Bps

Baud rate in Bps.

struct _flexio_i2c_master_transfer

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags

Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress

7-bit slave address.

flexio_i2c_direction_t direction

Transfer direction, read or write.

uint32_t subaddress

Sub address. Transferred MSB first.

uint8_t subaddressSize

Size of sub address.

uint8_t volatile *data

Transfer buffer.

volatile size_t dataSize

Transfer size.

struct _flexio_i2c_master_handle

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer

FlexIO I2C master transfer copy.

size_t transferSize

Total bytes to be transferred.

uint8_t state

Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback

Callback function called at transfer event. Callback function called at transfer event.

void *userData

Callback parameter passed to callback function.

bool needRestart

Whether master needs to send re-start signal.

2.24 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO_I2S_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in FLEXIO_I2S_Init(). Users may use the initialized structure unchanged in FLEXIO_I2S_Init() or modify some fields of the structure before calling FLEXIO_I2S_Init().

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXIO_I2S_Init to use the FlexIO I2S module.

Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO_I2S_Enable(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S module operation.

Parameters

- base – Pointer to FLEXIO_I2S_Type
- enable – True to enable, false dose not have any effect.

uint32_t FLEXIO_I2S_GetStatusFlags(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S status flags.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

Status flag, which are ORed by the enumerators in the `_flexio_i2s_status_flags`.

void FLEXIO_I2S_EnableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

```
void FLEXIO_I2S_DisableInterrupts(FLEXIO_I2S_Type *base, uint32_t mask)
```

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – pointer to *FLEXIO_I2S_Type* structure
- mask – interrupt source

```
static inline void FLEXIO_I2S_TxEnableDMA(FLEXIO_I2S_Type *base, bool enable)
```

Enables/disables the FlexIO I2S Tx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline void FLEXIO_I2S_RxEnableDMA(FLEXIO_I2S_Type *base, bool enable)
```

Enables/disables the FlexIO I2S Rx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

```
static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s send data register address.

```
static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(FLEXIO_I2S_Type *base)
```

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s receive data register address.

```
void FLEXIO_I2S_MasterSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format,  
                                uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – I2S master clock source frequency in Hz.

```
void FLEXIO_I2S_SlaveSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_format_t *format)
```

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.

```
status_t FLEXIO_I2S_WriteBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *txData,  
                                size_t size)
```

Sends data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

```
static inline void FLEXIO_I2S_WriteData(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint32_t  
                                       data)
```

Writes data into a data register.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- data – Data to be written.

```
status_t FLEXIO_I2S_ReadBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *rxData,  
                                size_t size)
```

Receives a piece of data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- rxData – Pointer to the data to be read.
- size – Bytes to be read.

Return values

- kStatus_Success – Successfully read data.

- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

static inline uint32_t FLEXIO_I2S_ReadData(*FLEXIO_I2S_Type* *base)

Reads a data from the data register.

Parameters

- base – FlexIO I2S base pointer

Returns

Data read from data register.

void FLEXIO_I2S_TransferTxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- handle – Pointer to *flexio_i2s_handle_t* structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

void FLEXIO_I2S_TransferSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

void FLEXIO_I2S_TransferRxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.
- handle – Pointer to *flexio_i2s_handle_t* structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.


```
status_t FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t
                                             *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXIO_I2S_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus_InvalidArgument – The input parameter is invalid.

```
status_t FLEXIO_I2S_TransferReceiveNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t
                                                *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data receive.
- kStatus_FLEXIO_I2S_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

```
void FLEXIO_I2S_TransferAbortSend(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)
```

Aborts the current send.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

void FLEXIO_I2S_TransferAbortReceive(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)
Aborts the current receive.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

status_t FLEXIO_I2S_TransferGetSendCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be sent.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes sent.

Return values

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

status_t FLEXIO_I2S_TransferGetReceiveCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *size_t* *count)

Gets the remaining bytes to be received.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes recieved.

Return values

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

Returns

count Bytes received.

void FLEXIO_I2S_TransferTxHandleIRQ(void *i2sBase, void *i2sHandle)

Tx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure

`void FLEXIO_I2S_TransferRxHandleIRQ(void *i2sBase, void *i2sHandle)`

Rx interrupt handler.

Parameters

- `i2sBase` – Pointer to `FLEXIO_I2S_Type` structure.
- `i2sHandle` – Pointer to `flexio_i2s_handle_t` structure.

`FSL_FLEXIO_I2S_DRIVER_VERSION`

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator `kStatus_FLEXIO_I2S_Idle`

FlexIO I2S is in idle state

enumerator `kStatus_FLEXIO_I2S_TxBusy`

FlexIO I2S Tx is busy

enumerator `kStatus_FLEXIO_I2S_RxBusy`

FlexIO I2S Rx is busy

enumerator `kStatus_FLEXIO_I2S_Error`

FlexIO I2S error occurred

enumerator `kStatus_FLEXIO_I2S_QueueFull`

FlexIO I2S transfer queue is full.

enumerator `kStatus_FLEXIO_I2S_Timeout`

FlexIO I2S timeout polling status flags.

enum `flexio_i2s_master_slave`

Master or slave mode.

Values:

enumerator `kFLEXIO_I2S_Master`

Master mode

enumerator `kFLEXIO_I2S_Slave`

Slave mode

`_flexio_i2s_interrupt_enable` Define FlexIO FlexIO I2S interrupt mask.

Values:

enumerator `kFLEXIO_I2S_TxDataRegEmptyInterruptEnable`

Transmit buffer empty interrupt enable.

enumerator `kFLEXIO_I2S_RxDataRegFullInterruptEnable`

Receive buffer full interrupt enable.

`_flexio_i2s_status_flags` Define FlexIO FlexIO I2S status mask.

Values:

enumerator `kFLEXIO_I2S_TxDataRegEmptyFlag`

Transmit buffer empty flag.

enumerator kFLEXIO_I2S_RxDataRegFullFlag
Receive buffer full flag.

enum _flexio_i2s_sample_rate
Audio sample rate.

Values:

enumerator kFLEXIO_I2S_SampleRate8KHz
Sample rate 8000Hz

enumerator kFLEXIO_I2S_SampleRate11025Hz
Sample rate 11025Hz

enumerator kFLEXIO_I2S_SampleRate12KHz
Sample rate 12000Hz

enumerator kFLEXIO_I2S_SampleRate16KHz
Sample rate 16000Hz

enumerator kFLEXIO_I2S_SampleRate22050Hz
Sample rate 22050Hz

enumerator kFLEXIO_I2S_SampleRate24KHz
Sample rate 24000Hz

enumerator kFLEXIO_I2S_SampleRate32KHz
Sample rate 32000Hz

enumerator kFLEXIO_I2S_SampleRate44100Hz
Sample rate 44100Hz

enumerator kFLEXIO_I2S_SampleRate48KHz
Sample rate 48000Hz

enumerator kFLEXIO_I2S_SampleRate96KHz
Sample rate 96000Hz

enum _flexio_i2s_word_width
Audio word width.

Values:

enumerator kFLEXIO_I2S_WordWidth8bits
Audio data width 8 bits

enumerator kFLEXIO_I2S_WordWidth16bits
Audio data width 16 bits

enumerator kFLEXIO_I2S_WordWidth24bits
Audio data width 24 bits

enumerator kFLEXIO_I2S_WordWidth32bits
Audio data width 32 bits

typedef struct _flexio_i2s_type FLEXIO_I2S_Type
Define FlexIO I2S access structure typedef.

typedef enum _flexio_i2s_master_slave flexio_i2s_master_slave_t
Master or slave mode.

typedef struct _flexio_i2s_config flexio_i2s_config_t
FlexIO I2S configure structure.

```
typedef struct _flexio_i2s_format flexio_i2s_format_t
    FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.
typedef enum _flexio_i2s_sample_rate flexio_i2s_sample_rate_t
    Audio sample rate.
typedef enum _flexio_i2s_word_width flexio_i2s_word_width_t
    Audio word width.
typedef struct _flexio_i2s_transfer flexio_i2s_transfer_t
    Define FlexIO I2S transfer structure.
typedef struct _flexio_i2s_handle flexio_i2s_handle_t
typedef void (*flexio_i2s_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,
status_t status, void *userData)
    FlexIO I2S xfer callback prototype.
I2S__RETRY_TIMES
    Retry times for waiting flag.
FLEXIO_I2S_XFER_QUEUE_SIZE
    FlexIO I2S transfer queue size, user can refine it according to use case.
struct _flexio_i2s_type
    #include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.
```

Public Members

```
FLEXIO_Type *flexioBase
    FlexIO base pointer
uint8_t txPinIndex
    Tx data pin index in FlexIO pins
uint8_t rxPinIndex
    Rx data pin index
uint8_t bclkPinIndex
    Bit clock pin index
uint8_t fsPinIndex
    Frame sync pin index
uint8_t txShifterIndex
    Tx data shifter index
uint8_t rxShifterIndex
    Rx data shifter index
uint8_t bclkTimerIndex
    Bit clock timer index
uint8_t fsTimerIndex
    Frame sync timer index
struct _flexio_i2s_config
    #include <fsl_flexio_i2s.h> FlexIO I2S configure structure.
```

Public Members

bool enableI2S

Enable FlexIO I2S

flexio_i2s_master_slave_t masterSlave

Master or slave

flexio_pin_polarity_t txPinPolarity

Tx data pin polarity, active high or low

flexio_pin_polarity_t rxPinPolarity

Rx data pin polarity

flexio_pin_polarity_t bclkPinPolarity

Bit clock pin polarity

flexio_pin_polarity_t fsPinPolarity

Frame sync pin polarity

flexio_shifter_timer_polarity_t txTimerPolarity

Tx data valid on bclk rising or falling edge

flexio_shifter_timer_polarity_t rxTimerPolarity

Rx data valid on bclk rising or falling edge

struct *_flexio_i2s_format*

#include <fsl_flexio_i2s.h> FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

Public Members

uint8_t bitWidth

Bit width of audio data, always 8/16/24/32 bits

uint32_t sampleRate_Hz

Sample rate of the audio data

struct *_flexio_i2s_transfer*

#include <fsl_flexio_i2s.h> Define FlexIO I2S transfer structure.

Public Members

uint8_t *data

Data buffer start pointer

size_t dataSize

Bytes to be transferred.

struct *_flexio_i2s_handle*

#include <fsl_flexio_i2s.h> Define FlexIO I2S handle structure.

Public Members

uint32_t state

Internal state

flexio_i2s_callback_t callback

Callback function called at transfer event

void *userData
 Callback parameter passed to callback function

uint8_t bitWidth
 Bit width for transfer, 8/16/24/32bits

flexio_i2s_transfer_t queue[(4U)]
 Transfer queue storing queued transfer

size_t transferSize[(4U)]
 Data bytes need to transfer

volatile uint8_t queueUser
 Index for user to queue transfer

volatile uint8_t queueDriver
 Index for driver to get the transfer data and size

2.25 FlexIO SPI Driver

```
void FLEXIO_SPI_MasterInit(FLEXIO_SPI_Type *base, flexio_spi_master_config_t
                           *masterConfig, uint32_t srcClock_Hz)
```

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
    .enableMaster = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 500000,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;  
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1. Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3. For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $3 \times 2 = 6$. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by $(1.5 + 2.5) \times 2 = 8$. Example

```
FLEXIO_SPI_Type spiDev = {  
    .flexioBase = FLEXIO,  
    .SDOPinIndex = 0,  
    .SDIPinIndex = 1,  
    .SCKPinIndex = 2,  
    .CSnPinIndex = 3,  
    .shifterIndex = {0,1},  
    .timerIndex = {0}  
};  
flexio_spi_slave_config_t config = {  
    .enableSlave = true,  
    .enableInDoze = false,  
    .enableInDebug = true,  
    .enableFastAccess = false,  
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,  
    .direction = kFLEXIO_SPI_MsbFirst,  
    .dataMode = kFLEXIO_SPI_8BitMode  
};  
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```


Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – status flag The parameter can be any combination of the following values:
 - kFLEXIO_SPI_TxEmptyFlag
 - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

```
void FLEXIO_SPI_DisableInterrupts(FLEXIO_SPI_Type *base, uint32_t mask)
```

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- mask – interrupt source The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

```
void FLEXIO_SPI_EnableDMA(FLEXIO_SPI_Type *base, uint32_t mask, bool enable)
```

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

```
static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_shift_direction_t  
                                                         direction)
```

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

```
static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_shift_direction_t  
                                                         direction)
```

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

```
static inline void FLEXIO_SPI_Enable(FLEXIO_SPI_Type *base, bool enable)
```

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type*.
- enable – True to enable, false does not have any effect.

```
void FLEXIO_SPI_MasterSetBaudRate(FLEXIO_SPI_Type *base, uint32_t baudRate_Bps,  
                                uint32_t srcClockHz)
```

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

```
static inline void FLEXIO_SPI_WriteData(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                       direction, uint32_t data)
```

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

```
static inline uint32_t FLEXIO_SPI_ReadData(FLEXIO_SPI_Type *base,  
                                           flexio_spi_shift_direction_t direction)
```

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

```
status_t FLEXIO_SPI_WriteBlocking(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                  direction, const uint8_t *buffer, size_t size)
```

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, *uint8_t* *buffer, *size_t* size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

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

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, *void* *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.

- `userData` – The parameter of the callback function.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`status_t` FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base,
flexio_spi_master_handle_t *handle,
flexio_spi_transfer_t *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- `base` – Pointer to the FLEXIO_SPI_Type structure.
- `handle` – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- `xfer` – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – SPI is not idle, is running another transfer.

`void` FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, flexio_spi_master_handle_t *handle)

Aborts the master data transfer, which used IRQ.

Parameters

- `base` – Pointer to the FLEXIO_SPI_Type structure.
- `handle` – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

`status_t` FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base,
flexio_spi_master_handle_t *handle, size_t *count)

Gets the data transfer status which used IRQ.

Parameters

- `base` – Pointer to the FLEXIO_SPI_Type structure.
- `handle` – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

`void` FLEXIO_SPI_MasterTransferHandleIRQ(`void` *spiType, `void` *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- `spiType` – Pointer to the FLEXIO_SPI_Type structure.

- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

```
status_t FLEXIO_SPI_SlaveTransferCreateHandle(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle,  
                                              flexio_spi_slave_transfer_callback_t callback,  
                                              void *userData)
```

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus__Success – Successfully create the handle.
- kStatus__OutOfRange – The FlexIO type/handle/ISR table out of range.

```
status_t FLEXIO_SPI_SlaveTransferNonBlocking(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle,  
                                              flexio_spi_transfer_t *xfer)
```

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus__Success – Successfully start a transfer.
- kStatus__InvalidArgument – Input argument is invalid.
- kStatus__FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbort(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle)
```

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCount(FLEXIO_SPI_Type *base,  
                                              flexio_spi_slave_handle_t *handle,  
                                              size_t *count)
```

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)`

FlexIO SPI slave IRQ handler function.

Parameters

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.
- `spiHandle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

`FSL_FLEXIO_SPI_DRIVER_VERSION`

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator `kStatus_FLEXIO_SPI_Busy`

FlexIO SPI is busy.

enumerator `kStatus_FLEXIO_SPI_Idle`

SPI is idle

enumerator `kStatus_FLEXIO_SPI_Error`

FlexIO SPI error.

enumerator `kStatus_FLEXIO_SPI_Timeout`

FlexIO SPI timeout polling status flags.

enum `_flexio_spi_clock_phase`

FlexIO SPI clock phase configuration.

Values:

enumerator `kFLEXIO_SPI_ClockPhaseFirstEdge`

First edge on SPCK occurs at the middle of the first cycle of a data transfer.

enumerator `kFLEXIO_SPI_ClockPhaseSecondEdge`

First edge on SPCK occurs at the start of the first cycle of a data transfer.

enum `_flexio_spi_shift_direction`

FlexIO SPI data shifter direction options.

Values:

enumerator `kFLEXIO_SPI_MsbFirst`

Data transfers start with most significant bit.

enumerator `kFLEXIO_SPI_LsbFirst`

Data transfers start with least significant bit.

enum `_flexio_spi_data_bitcount_mode`

FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode
8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags
Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag
Receive buffer full flag.

enum _flexio_spi_dma_enable
Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable
Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable
Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable
All DMA request source

enum _flexio_spi_transfer_flags
Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb
FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb
FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb
FlexIO SPI 16-bit MSB first


```

enumerator kFLEXIO_SPI_16bitLsb
    FlexIO SPI 16-bit LSB first
enumerator kFLEXIO_SPI_32bitMsb
    FlexIO SPI 32-bit MSB first
enumerator kFLEXIO_SPI_32bitLsb
    FlexIO SPI 32-bit LSB first
enumerator kFLEXIO_SPI_csContinuous
    Enable the CS signal continuous mode
typedef enum _flexio_spi_clock_phase flexio_spi_clock_phase_t
    FlexIO SPI clock phase configuration.
typedef enum _flexio_spi_shift_direction flexio_spi_shift_direction_t
    FlexIO SPI data shifter direction options.
typedef enum _flexio_spi_data_bitcount_mode flexio_spi_data_bitcount_mode_t
    FlexIO SPI data length mode options.
typedef struct _flexio_spi_type FLEXIO_SPI_Type
    Define FlexIO SPI access structure typedef.
typedef struct _flexio_spi_master_config flexio_spi_master_config_t
    Define FlexIO SPI master configuration structure.
typedef struct _flexio_spi_slave_config flexio_spi_slave_config_t
    Define FlexIO SPI slave configuration structure.
typedef struct _flexio_spi_transfer flexio_spi_transfer_t
    Define FlexIO SPI transfer structure.
typedef struct _flexio_spi_master_handle flexio_spi_master_handle_t
    typedef for flexio_spi_master_handle_t in advance.
typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t
    Slave handle is the same with master handle.
typedef void (*flexio_spi_master_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_master_handle_t *handle, status_t status, void *userData)
    FlexIO SPI master callback for finished transmit.
typedef void (*flexio_spi_slave_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_slave_handle_t *handle, status_t status, void *userData)
    FlexIO SPI slave callback for finished transmit.
FLEXIO_SPI_DUMMYDATA
    FlexIO SPI dummy transfer data, the data is sent while txData is NULL.
SPI_RETRY_TIMES
    Retry times for waiting flag.
FLEXIO_SPI_XFER_DATA_FORMAT(flag)
    Get the transfer data format of width and bit order.
struct _flexio_spi_type
    #include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

```

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

uint8_t SDOPinIndex

Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex

Pin select for data input.

uint8_t SCKPinIndex

Pin select for clock.

uint8_t CSnPinIndex

Pin select for enable.

uint8_t shifterIndex[2]

Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]

Timer index used in FlexIO SPI.

struct _flexio_spi_master_config

#include <fsl_flexio_spi.h> Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster

Enable/disable FlexIO SPI master after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct _flexio_spi_slave_config

#include <fsl_flexio_spi.h> Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave

Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase

Clock phase.

flexio_spi_data_bitcount_mode_t dataMode

8bit or 16bit mode.

struct *_flexio_spi_transfer*

#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

size_t dataSize

Transfer bytes.

uint8_t flags

FlexIO SPI control flag, MSB first or LSB first.

struct *_flexio_spi_master_handle*

#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData

Transfer buffer.

uint8_t *rxData

Receive buffer.

size_t transferSize

Total bytes to be transferred.

volatile size_t txRemainingBytes

Send data remaining in bytes.

volatile size_t rxRemainingBytes

Receive data remaining in bytes.

volatile uint32_t state

FlexIO SPI internal state.

uint8_t bytePerFrame

SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction

Shift direction.

flexio_spi_master_transfer_callback_t callback

FlexIO SPI callback.

void *userData

Callback parameter.

bool isCsContinuous

Is current transfer using CS continuous mode.

uint32_t timer1Cfg

TIMER1 TIMCFG regiser value backup.

2.26 FlexIO UART Driver

status_t FLEXIO_UART_Init(*FLEXIO_UART_Type* *base, const *flexio_uart_config_t* *userConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_GetDefaultConfig().

Example

```
FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
    .bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXIO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_UART_TxDataRegEmptyFlag
 - kFLEXIO_UART_RxEmptyFlag
 - kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

```
static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(FLEXIO_UART_Type *base)
```

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART receive data register address.

```
static inline void FLEXIO_UART_EnableTxDMA(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_EnableRxDMA(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

```
static inline void FLEXIO_UART_Enable(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

```
static inline void FLEXIO_UART_WriteByte(FLEXIO_UART_Type *base, const uint8_t *buffer)
```

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

```
static inline void FLEXIO_UART_ReadByte(FLEXIO_UART_Type *base, uint8_t *buffer)
```

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.

- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

```
void FLEXIO_UART_TransferStartRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle, uint8_t *ringBuffer, size_t
                                         ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

```
void FLEXIO_UART_TransferStopRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

```
status_t FLEXIO_UART_TransferSendNonBlocking(FLEXIO_UART_Type *base,
                                              flexio_uart_handle_t *handle,
                                              flexio_uart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `xfer` – FlexIO UART transfer structure. See `flexio_uart_transfer_t`.

Return values

- `kStatus_Success` – Successfully starts the data transmission.
- `kStatus_UART_TxBusy` – Previous transmission still not finished, data not written to the TX register.

```
void FLEXIO_UART_TransferAbortSend(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
```

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the `remainBytes` to find out how many bytes are still not sent out.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetSendCount(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `count` – Number of bytes sent so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

```
status_t FLEXIO_UART_TransferReceiveNonBlocking(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer, size_t *receivedBytes)
```

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to `xfer->data`. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the

xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See flexio_uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus__Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

void FLEXIO_UART_TransferAbortReceive(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetReceiveCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *size_t* *count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus__Success – Successfully return the count.

void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy

Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy

Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle

UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR

ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun

UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout

UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport

Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char

FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar

7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar

8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar

9-bit data characters

enum _flexio_uart_interrupt_enable

Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
Define FlexIO UART status mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag
Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag
Receive buffer over run flag.

typedef enum _flexio_uart_bit_count_per_char flexio_uart_bit_count_per_char_t
FlexIO UART bit count per char.

typedef struct _flexio_uart_type FLEXIO_UART_Type
Define FlexIO UART access structure typedef.

typedef struct _flexio_uart_config flexio_uart_config_t
Define FlexIO UART user configuration structure.

typedef struct _flexio_uart_transfer flexio_uart_transfer_t
Define FlexIO UART transfer structure.

typedef struct _flexio_uart_handle flexio_uart_handle_t

typedef void (*flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, status_t status, void *userData)
FlexIO UART transfer callback function.

UART_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_uart_type
#include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t TxPinIndex
Pin select for UART_Tx.

uint8_t RxPinIndex
Pin select for UART_Rx.

uint8_t shifterIndex[2]
Shifter index used in FlexIO UART.

uint8_t timerIndex[2]
Timer index used in FlexIO UART.

struct _flexio_uart_config
#include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

Public Members

`bool enableUart`
Enable/disable FlexIO UART TX & RX.

`bool enableInDoze`
Enable/disable FlexIO operation in doze mode

`bool enableInDebug`
Enable/disable FlexIO operation in debug mode

`bool enableFastAccess`
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

`uint32_t baudRate_Bps`
Baud rate in Bps.

`flexio_uart_bit_count_per_char_t bitCountPerChar`
number of bits, 7/8/9 -bit

`struct _flexio_uart_transfer`
#include <fsl_flexio_uart.h> Define FlexIO UART transfer structure.

Public Members

`size_t dataSize`
Transfer size

`struct _flexio_uart_handle`
#include <fsl_flexio_uart.h> Define FLEXIO UART handle structure.

Public Members

`const uint8_t *volatile txData`
Address of remaining data to send.

`volatile size_t txDataSize`
Size of the remaining data to send.

`uint8_t *volatile rxData`
Address of remaining data to receive.

`volatile size_t rxDataSize`
Size of the remaining data to receive.

`size_t txDataSizeAll`
Total bytes to be sent.

`size_t rxDataSizeAll`
Total bytes to be received.

`uint8_t *rxRingBuffer`
Start address of the receiver ring buffer.

`size_t rxRingBufferSize`
Size of the ring buffer.

`volatile uint16_t rxRingBufferHead`
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
Callback function.

void *userData
UART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

union ____unnamed79____

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.

2.27 FREQME: Frequency Measurement

2.28 GDET

status_t GDET__Init(GDET_Type *base)
Initialize GDET.
This function initializes GDET block and setting.

Parameters

- base – GDET peripheral base address

Returns

Status of the init operation

void GDET__Deinit(GDET_Type *base)
Deinitialize GDET.
This function deinitializes GDET secure counter.

Parameters

- base – GDET peripheral base address

status_t GDET__Enable(GDET_Type *base)
Enable GDET.
This function enables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns

Status of the enable operation

status_t GDET_Disable(GDET_Type *base)

Disable GDET.

This function disables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns

Status of the disable operation

status_t GDET_IsolateOn(GDET_Type *base)

Turn on GDET isolation.

This function turns on isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns

Status of the operation

status_t GDET_IsolateOff(GDET_Type *base)

Turn off GDET isolation.

This function turns off isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns

Status of the operation

status_t GDET_ReconfigureVoltageMode(GDET_Type *base, *gdet_core_voltage_t* voltage)

Change expected core voltage.

This function changes core voltage which Glitch detector expect.

Parameters

- base – GDET peripheral base address
- voltage – Expected core voltage

Returns

Status of the GDET reconfiguration operation

FSL_GDET_DRIVER_VERSION

Defines GDET driver version 2.1.0.

Change log:

- Version 2.1.0
 - Update for multiple instances
 - Fix bug in isolation off API
 - Add enable and disable APIs
- 2.0.1
 - Fix MISRA in GDET_ReconfigureVoltageMode()

- Version 2.0.0
 - initial version

typedef uint32_t gdet_core_voltage_t

GDET Core Voltage.

These constants are used to define core voltage argument to be used with GDET_ReconfigureVoltageMode(). Different SoC may support various volatages, refer to documentation.

void GDET_DriverIRQHandler(void)

kGDET_0_9v

Voltage (0.9V)

kGDET_1_0v

Voltage (1.0V)

kGDET_1_1v

Voltage (1.1V)

2.29 Glitch Detect

2.30 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION

GPIO driver version.

enum _gpio_pin_direction

GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput

Set current pin as digital input

enumerator kGPIO_DigitalOutput

Set current pin as digital output

enum _gpio_checker_attribute

GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW

User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW

User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW

User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW

User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW

User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO__UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO__UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO__UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO__IgnoreAttributeCheck
Ignores the attribute check

enum __gpio_interrupt_config
Configures the interrupt generation condition.

Values:

enumerator kGPIO__InterruptStatusFlagDisabled
Interrupt status flag is disabled.

enumerator kGPIO__DMARisingEdge
ISF flag and DMA request on rising edge.

enumerator kGPIO__DMAFallingEdge
ISF flag and DMA request on falling edge.

enumerator kGPIO__DMAEitherEdge
ISF flag and DMA request on either edge.

enumerator kGPIO__FlagRisingEdge
Flag sets on rising edge.

enumerator kGPIO__FlagFallingEdge
Flag sets on falling edge.

enumerator kGPIO__FlagEitherEdge
Flag sets on either edge.

enumerator kGPIO__InterruptLogicZero
Interrupt when logic zero.

enumerator kGPIO__InterruptRisingEdge
Interrupt on rising edge.

enumerator kGPIO__InterruptFallingEdge
Interrupt on falling edge.

enumerator kGPIO__InterruptEitherEdge
Interrupt on either edge.

enumerator kGPIO__InterruptLogicOne
Interrupt when logic one.

enumerator kGPIO__ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kGPIO__ActiveLowTriggerOutputEnable
Enable active low-trigger output.

enum __gpio_interrupt_selection
Configures the selection of interrupt/DMA request/trigger output.

Values:

enumerator kGPIO_InterruptOutput0
Interrupt/DMA request/trigger output 0.

enumerator kGPIO_InterruptOutput1
Interrupt/DMA request/trigger output 1.

enum gpio_pin_interrupt_control_t
GPIO pin and interrupt control.

Values:

enumerator kGPIO_PinControlNonSecure
Pin Control Non-Secure.

enumerator kGPIO_InterruptControlNonSecure
Interrupt Control Non-Secure.

enumerator kGPIO_PinControlNonPrivilege
Pin Control Non-Privilege.

enumerator kGPIO_InterruptControlNonPrivilege
Interrupt Control Non-Privilege.

typedef enum _gpio_pin_direction gpio_pin_direction_t
GPIO direction definition.

typedef enum _gpio_checker_attribute gpio_checker_attribute_t
GPIO checker attribute.

typedef struct _gpio_pin_config gpio_pin_config_t
The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

typedef enum _gpio_interrupt_config gpio_interrupt_config_t
Configures the interrupt generation condition.

typedef enum _gpio_interrupt_selection gpio_interrupt_selection_t
Configures the selection of interrupt/DMA request/trigger output.

typedef struct _gpio_version_info gpio_version_info_t
GPIO version information.

GPIO_FIT_REG(value)

struct _gpio_pin_config
#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Public Members

gpio_pin_direction_t pinDirection
GPIO direction, input or output

uint8_t outputLogic
Set a default output logic, which has no use in input

struct _gpio_version_info
#include <fsl_gpio.h> GPIO version information.

Public Members

uint16_t feature

Feature Specification Number.

uint8_t minor

Minor Version Number.

uint8_t major

Major Version Number.

2.31 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PortDeinit(GPIO_Type *base)

Deinitializes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

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, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or an 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 (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

void GPIO_GetVersionInfo(GPIO_Type *base, *gpio_version_info_t* *info)

Get GPIO version information.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

- info – GPIO version information

```
static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, gpio_pin_interrupt_control_t mask)
```

lock or unlock secure privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

```
static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Enable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Disable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
```

Enable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
```

Disable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Enable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_DisableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
```

Disable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline void GPIO_EnableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
```

Enable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Disable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputEnable(GPIO_Type *base, uint32_t mask)

Enable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputDisable(GPIO_Type *base, uint32_t mask)

Disable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- 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 void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

```
static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)
```

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

```
static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin,  
                                             gpio_interrupt_config_t config)
```

Configures the gpio pin interrupt/DMA request.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- config – GPIO pin interrupt configuration.
 - kGPIO_InterruptStatusFlagDisabled: Interrupt/DMA request disabled.
 - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
 - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
 - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
 - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
 - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
 - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
 - kGPIO_InterruptLogicZero : Interrupt when logic zero.
 - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
 - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
 - kGPIO_InterruptEitherEdge : Interrupt on either edge.
 - kGPIO_InterruptLogicOne : Interrupt when logic one.
 - kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
 - kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

```
static inline void GPIO_SetPinInterruptChannel(GPIO_Type *base, uint32_t pin,  
                                              gpio_interrupt_selection_t selection)
```

Configures the gpio pin interrupt/DMA request/trigger output channel selection.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- selection – GPIO pin interrupt output selection.

- kGPIO_InterruptOutput0: Interrupt/DMA request/trigger output 0.
- kGPIO_InterruptOutput1 : Interrupt/DMA request/trigger output 1.

uint32_t GPIO_GpioGetInterruptFlags(GPIO_Type *base)

Read the GPIO interrupt status flags.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)

Returns

The current GPIO's interrupt status flag. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint32_t GPIO_GpioGetInterruptChannelFlags(GPIO_Type *base, uint32_t channel)

Read the GPIO interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

Returns

The current GPIO's interrupt status flag based on the selected interrupt channel. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint8_t GPIO_PinGetInterruptFlag(GPIO_Type *base, uint32_t pin)

Read individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on)
- pin – GPIO specific pin number.

Returns

The current selected pin's interrupt status flag.

void GPIO_GpioClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

void GPIO_GpioClearInterruptChannelFlags(GPIO_Type *base, uint32_t mask, uint32_t channel)

Clears GPIO pin interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

void GPIO_PinClearInterruptFlag(GPIO_Type *base, uint32_t pin)

Clear GPIO individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on).
- pin – GPIO specific pin number.

```
static inline void GPIO_SetMultipleInterruptPinsConfig(GPIO_Type *base, uint32_t mask,  
                                                    gpio_interrupt_config_t config)
```

Sets the GPIO interrupt configuration in PCR register for multiple pins.

Parameters

- base – GPIO peripheral base pointer.
- mask – GPIO pin number macro.
- config – GPIO pin interrupt configuration.
 - kGPIO_InterruptStatusFlagDisabled: Interrupt disabled.
 - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
 - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
 - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
 - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
 - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
 - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
 - kGPIO_InterruptLogicZero : Interrupt when logic zero.
 - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
 - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
 - kGPIO_InterruptEitherEdge : Interrupt on either edge.
 - kGPIO_InterruptLogicOne : Interrupt when logic one.
 - kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
 - kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

```
void GPIO_CheckAttributeBytes(GPIO_Type *base, gpio_checker_attribute_t attribute)
```

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

2.32 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.

2.33 I3C Common Driver

```
typedef struct _i3c_config i3c_config_t
```

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
uint32_t I3C_GetInstance(I3C_Type *base)
```

Get which instance current I3C is used.

Parameters

- base – The I3C peripheral base address.

```
void I3C_GetDefaultConfig(i3c_config_t *config)
```

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster      = kI3C_MasterCapable;
config->disableTimeout    = false;
config->hKeep              = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop = true;
config->enableOpenDrainHigh = true;
config->baudRate_Hz.i2cBaud = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress = 0x0AU;
config->slowClock_Hz       = 1000000U;
config->enableSlave        = true;
config->vendorID            = 0x11BU;
config->enableRandomPart    = false;
config->partNumber          = 0;
config->dcr                 = 0;
config->bcr = 0;
config->hdrMode             = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest        = false;
config->ignoreS0S1Error     = false;
config->offline              = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C_Init().

Parameters

- config – **[out]** User provided configuration structure for default values. Refer to i3c_config_t.

```
void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)
```

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.

- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`struct __i3c_config`

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

`i3c_master_enable_t` enableMaster

Enable master mode.

`bool` disableTimeout

Whether to disable timeout to prevent the ERRWARN.

`i3c_master_hkeep_t` hKeep

High keeper mode setting.

`bool` enableOpenDrainStop

Whether to emit open-drain speed STOP.

`bool` enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

`i3c_baudrate_hz_t` baudRate_Hz

Desired baud rate settings.

`i3c_start_scl_delay_t` startSclDelay

I3C SCL delay after START.

`i3c_start_scl_delay_t` restartSclDelay

I3C SCL delay after Repeated START.

`uint8_t` masterDynamicAddress

Main master dynamic address configuration.

`uint32_t` maxWriteLength

Maximum write length.

`uint32_t` maxReadLength

Maximum read length.

`bool` enableSlave

Whether to enable slave.

`uint8_t` staticAddr

Static address.

`uint16_t` vendorID

Device vendor ID(manufacture ID).

`uint32_t` partNumber

Device part number info

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint8_t hdrMode

Support hdr mode, could be OR logic in enumeration: `i3c_hdr_mode_t`.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

2.34 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.

`i3c_start_scl_delay_t startSclDelay`

I3C SCL delay after START.

`i3c_start_scl_delay_t restartSclDelay`

I3C SCL delay after Repeated START.

`struct __i3c_master_transfer_callback`

#include <fsl_i3c.h> i3c master callback functions.

Public Members

`void (*slave2Master)(I3C_Type *base, void *userData)`

Transfer complete callback

`void (*ibiCallback)(I3C_Type *base, i3c_master_handle_t *handle, i3c_ibi_type_t ibiType, i3c_ibi_state_t ibiState)`

IBI event callback

`void (*transferComplete)(I3C_Type *base, i3c_master_handle_t *handle, status_t completionStatus, void *userData)`

Transfer complete callback

`struct __i3c_master_transfer`

#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the `I3C_MasterTransferNonBlocking()` API.

Public Members

`uint32_t flags`
Bit mask of options for the transfer. See enumeration `_i3c_master_transfer_flags` for available options. Set to 0 or `kI3C_TransferDefaultFlag` for normal transfers.

`uint8_t slaveAddress`
The 7-bit slave address.

`i3c_direction_t direction`
Either `kI3C_Read` or `kI3C_Write`.

`uint32_t subaddress`
Sub address. Transferred MSB first.

`size_t subaddressSize`
Length of sub address to send in bytes. Maximum size is 4 bytes.

`void *data`
Pointer to data to transfer.

`size_t dataSize`
Number of bytes to transfer.

`i3c_bus_type_t busType`
bus type.

`i3c_ibi_response_t ibiResponse`
ibi response during transfer.

`struct _i3c_master_handle`
`#include <fsl_i3c.h>` Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

`uint8_t state`
Transfer state machine current state.

`uint32_t remainingBytes`
Remaining byte count in current state.

`i3c_rx_term_ops_t rxTermOps`
Read termination operation.

`i3c_master_transfer_t transfer`
Copy of the current transfer info.

`uint8_t ibiAddress`
Slave address which request IBI.

`uint8_t *ibiBuff`
Pointer to IBI buffer to keep ibi bytes.

`size_t ibiPayloadSize`
IBI payload size.

`i3c_ibi_type_t ibiType`
IBI type.

i3c_master_transfer_callback_t callback

Callback functions pointer.

void *userData

Application data passed to callback.

2.35 I3C Master DMA Driver

```
void I3C_MasterTransferCreateHandleEDMA(I3C_Type *base, i3c_master_edma_handle_t
                                         *handle, const i3c_master_edma_callback_t
                                         *callback, void *userData, edma_handle_t
                                         *rxDmaHandle, edma_handle_t *txDmaHandle)
```

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.
- rxDmaHandle – Handle for the DMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_MasterTransferEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle,
                                i3c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t I3C_MasterTransferGetCountEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle,
                                         size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.

- `handle` – Pointer to the I3C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

`void I3C_MasterTransferAbortEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)`

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `i3cHandle` – Pointer to the I3C master DMA driver handle.

`typedef struct _i3c_master_edma_handle i3c_master_edma_handle_t`

`typedef struct _i3c_master_edma_callback i3c_master_edma_callback_t`
i3c master callback functions.

`struct _i3c_master_edma_callback`
`#include <fsl_i3c_edma.h>` i3c master callback functions.

Public Members

`void (*slave2Master)(I3C_Type *base, void *userData)`

Target asks for controller request.

`void (*ibiCallback)(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_ibi_type_t ibiType, i3c_ibi_state_t ibiState)`

IBI event callback.

`void (*transferComplete)(I3C_Type *base, i3c_master_edma_handle_t *handle, status_t status, void *userData)`

Transfer complete callback.


```
struct _i3c_master_edma_handle
#include <fsl_i3c_edma.h> Driver handle for master EDMA APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

I3C_Type *base
I3C base pointer.

uint8_t state
Transfer state machine current state.

uint32_t transferCount
Indicates progress of the transfer

uint8_t subaddressBuffer[4]
Saving subaddress command.

uint8_t subaddressCount
Saving command count.

i3c_master_transfer_t transfer
Copy of the current transfer info.

i3c_master_edma_callback_t callback
Callback function pointer.

void *userData
Application data passed to callback.

edma_handle_t *rxDmaHandle
Handle for receive DMA channel.

edma_handle_t *txDmaHandle
Handle for transmit DMA channel.

bool ibiFlag
IBIWON flag.

uint8_t ibiAddress
Slave address which request IBI.

uint8_t *ibiBuff
Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
IBI payload size.

i3c_ibi_type_t ibiType
IBI type.

status_t result
Transfer result.

2.36 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_SlaveErrorOvrerrunFlag

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

2.37 I3C Slave DMA Driver

```
void I3C_SlaveTransferCreateHandleEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,
                                       i3c_slave_edma_callback_t callback, void *userData,
                                       edma_handle_t *rxDmaHandle, edma_handle_t
                                       *txDmaHandle)
```

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *I3C_SlaveTransferAbortDMA()* API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.

- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.
- `rxDmaHandle` – Handle for the DMA receive channel. Created by the user prior to calling this function.
- `txDmaHandle` – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_SlaveTransferEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,  
                             i3c_slave_edma_transfer_t *transfer, uint32_t eventMask)
```

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C slave driver handle.
- `transfer` – The pointer to the transfer descriptor.
- `eventMask` – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.
- `kStatus_Fail` – The transaction can't be set.

```
void I3C_SlaveTransferAbortEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle)
```

Abort a slave edma non-blocking transfer in a early time.

Parameters

- `base` – I3C peripheral base address
- `handle` – pointer to `i3c_slave_edma_handle_t` structure

```
void I3C_SlaveTransferEDMAHandleIRQ(I3C_Type *base, void *i3CHandle)
```

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `i3CHandle` – Pointer to the I3C slave DMA driver handle.

```
typedef struct _i3c_slave_edma_handle i3c_slave_edma_handle_t
```

```
typedef struct _i3c_slave_edma_transfer i3c_slave_edma_transfer_t  
I3C slave transfer structure.
```

```
typedef void (*i3c_slave_edma_callback_t)(I3C_Type *base, i3c_slave_edma_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

Param base

Base address for the I3C instance on which the event occurred.

Param handle

Pointer to slave DMA transfer handle.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _i3c_slave_edma_transfer
```

```
#include <fsl_i3c_edma.h> I3C slave transfer structure.
```

Public Members

```
uint32_t event
```

Reason the callback is being invoked.

```
uint8_t *txData
```

Transfer buffer

```
size_t txDataSize
```

Transfer size

```
uint8_t *rxData
```

Transfer buffer

```
size_t rxDataSize
```

Transfer size

```
status_t completionStatus
```

Success or error code describing how the transfer completed. Only applies for kI3C_SlaveCompletionEvent.

```
struct _i3c_slave_edma_handle
```

```
#include <fsl_i3c_edma.h> I3C slave edma handle structure.
```

Note: The contents of this structure are private and subject to change.

Public Members

```
I3C_Type *base
```

I3C base pointer.

```
i3c_slave_edma_transfer_t transfer
```

I3C slave transfer copy.

```
bool isBusy
```

Whether transfer is busy.

`bool` `wasTransmit`
Whether the last transfer was a transmit.

`uint32_t` `eventMask`
Mask of enabled events.

`i3c_slave_edma_callback_t` `callback`
Callback function called at transfer event.

`edma_handle_t` `*rxDmaHandle`
Handle for receive DMA channel.

`edma_handle_t` `*txDmaHandle`
Handle for transmit DMA channel.

`void` `*userData`
Callback parameter passed to callback.

2.38 INPUTMUX: Input Multiplexing Driver

`enum` `_inputmux_connection_t`
INPUTMUX connections type.

Values:

`enumerator` `kINPUTMUX_CtimerInp0ToTimer0Captsel`
TIMER0 CAPTSEL.

`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__CtimerInp18ToTimer0Cptsel
enumerator kINPUTMUX__CtimerInp19ToTimer0Cptsel
enumerator kINPUTMUX__Usb0StartOfFrameToTimer0Cptsel
enumerator kINPUTMUX__Usb1StartOfFrameToTimer0Cptsel
enumerator kINPUTMUX__DcdcBurstActiveToTimer0Cptsel
enumerator kINPUTMUX__Sai0TxSyncOutToTimer0Cptsel
enumerator kINPUTMUX__Sai0RxSyncOutToTimer0Cptsel
enumerator kINPUTMUX__Adc0IrqToTimer0Cptsel
enumerator kINPUTMUX__Adc1IrqToTimer0Cptsel
enumerator kINPUTMUX__Cmp0OutToTimer0Cptsel
enumerator kINPUTMUX__Cmp1OutToTimer0Cptsel
enumerator kINPUTMUX__Pwm0A0Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Pwm0A1Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Pwm0A2Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Pwm0A3Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Pwm1A0Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Pwm1A1Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Pwm1A2Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Pwm1A3Trig01ToTimer0Cptsel
enumerator kINPUTMUX__Qdc0CmpPosMatchToTimer0Cptsel
enumerator kINPUTMUX__Qdc1CmpPosMatchToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut0AToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut0BToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut1AToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut1BToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut2AToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut2BToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut3AToTimer0Cptsel
enumerator kINPUTMUX__EvtgOut3BToTimer0Cptsel
enumerator kINPUTMUX__LpFlexcomm0Trig0ToTimer0Cptsel
enumerator kINPUTMUX__LpFlexcomm0Trig1ToTimer0Cptsel
enumerator kINPUTMUX__LpFlexcomm0Trig2ToTimer0Cptsel
enumerator kINPUTMUX__LpFlexcomm1Trig0ToTimer0Cptsel

enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer0Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer0Cptsel
enumerator kINPUTMUX_Sai1TxSyncOutToTimer0Cptsel
enumerator kINPUTMUX_Sai1RxSyncOutToTimer0Cptsel
TIMER1 CAPTSEL.
enumerator kINPUTMUX_CtimerInp0ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp1ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp2ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp3ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp4ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp5ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp6ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp7ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp8ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp9ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp10ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp11ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp12ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp13ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp14ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp15ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp16ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp17ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp18ToTimer1Cptsel
enumerator kINPUTMUX_CtimerInp19ToTimer1Cptsel
enumerator kINPUTMUX_Usb0StartOfFrameToTimer1Cptsel

enumerator kINPUTMUX_Usb1StartOfFrameToTimer1Captsel
enumerator kINPUTMUX_DcdcBurstActiveToTimer1Captsel
enumerator kINPUTMUX_Sai0TxSyncOutToTimer1Captsel
enumerator kINPUTMUX_Sai0RxSyncOutToTimer1Captsel
enumerator kINPUTMUX_Adc0IrqToTimer1Captsel
enumerator kINPUTMUX_Adc1IrqToTimer1Captsel
enumerator kINPUTMUX_Cmp0OutToTimer1Captsel
enumerator kINPUTMUX_Cmp1OutToTimer1Captsel
enumerator kINPUTMUX_Pwm0A0Trig01ToTimer1Captsel
enumerator kINPUTMUX_Pwm0A1Trig01ToTimer1Captsel
enumerator kINPUTMUX_Pwm0A2Trig01ToTimer1Captsel
enumerator kINPUTMUX_Pwm0A3Trig01ToTimer1Captsel
enumerator kINPUTMUX_Pwm1A0Trig01ToTimer1Captsel
enumerator kINPUTMUX_Pwm1A1Trig01ToTimer1Captsel
enumerator kINPUTMUX_Pwm1A2Trig01ToTimer1Captsel
enumerator kINPUTMUX_Pwm1A3Trig01ToTimer1Captsel
enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer1Captsel
enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer1Captsel
enumerator kINPUTMUX_EvtgOut0AToTimer1Captsel
enumerator kINPUTMUX_EvtgOut0BToTimer1Captsel
enumerator kINPUTMUX_EvtgOut1AToTimer1Captsel
enumerator kINPUTMUX_EvtgOut1BToTimer1Captsel
enumerator kINPUTMUX_EvtgOut2AToTimer1Captsel
enumerator kINPUTMUX_EvtgOut2BToTimer1Captsel
enumerator kINPUTMUX_EvtgOut3AToTimer1Captsel
enumerator kINPUTMUX_EvtgOut3BToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer1Captsel

```
enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer1Captsel
enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer1Captsel
enumerator kINPUTMUX_Sai1TxSyncOutToTimer1Captsel
enumerator kINPUTMUX_Sai1RxSyncOutToTimer1Captsel
    TIMER2 CAPTSEL.
enumerator kINPUTMUX_CtimerInp0ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp1ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp2ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp3ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp4ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp5ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp6ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp7ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp8ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp9ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp10ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp11ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp12ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp13ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp14ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp15ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp16ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp17ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp18ToTimer2Captsel
enumerator kINPUTMUX_CtimerInp19ToTimer2Captsel
enumerator kINPUTMUX_Usb0StartOffFrameToTimer2Captsel
enumerator kINPUTMUX_Usb1StartOffFrameToTimer2Captsel
enumerator kINPUTMUX_DcdcBurstActiveToTimer2Captsel
enumerator kINPUTMUX_Sai0TxSyncOutToTimer2Captsel
```

enumerator kINPUTMUX__Sai0RxSyncOutToTimer2Cptsel
enumerator kINPUTMUX__Adc0IrqToTimer2Cptsel
enumerator kINPUTMUX__Adc1IrqToTimer2Cptsel
enumerator kINPUTMUX__Cmp0OutToTimer2Cptsel
enumerator kINPUTMUX__Cmp1OutToTimer2Cptsel
enumerator kINPUTMUX__Pwm0A0Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm0A1Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm0A2Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm0A3Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1A0Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1A1Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1A2Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Pwm1A3Trig01ToTimer2Cptsel
enumerator kINPUTMUX__Qdc0CmpPosMatchToTimer2Cptsel
enumerator kINPUTMUX__Qdc1CmpPosMatchToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut0AToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut0BToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut1AToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut1BToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut2AToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut2BToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut3AToTimer2Cptsel
enumerator kINPUTMUX__EvtgOut3BToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm0Trig0ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm0Trig1ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm0Trig2ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm1Trig0ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm1Trig1ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm1Trig2ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm2Trig0ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm2Trig1ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm2Trig2ToTimer2Cptsel
enumerator kINPUTMUX__LpFlexcomm3Trig0ToTimer2Cptsel

enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer2Captsel
enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer2Captsel
enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer2Captsel
enumerator kINPUTMUX_Sai1TxSyncOutToTimer2Captsel
enumerator kINPUTMUX_Sai1RxSyncOutToTimer2Captsel
TIMER3 CAPTSEL.
enumerator kINPUTMUX_CtimerInp0ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp1ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp2ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp3ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp4ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp5ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp6ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp7ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp8ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp9ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp10ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp11ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp12ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp13ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp14ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp15ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp16ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp17ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp18ToTimer3Captsel
enumerator kINPUTMUX_CtimerInp19ToTimer3Captsel
enumerator kINPUTMUX_Usb0StartOfFrameToTimer3Captsel
enumerator kINPUTMUX_Usb1StartOfFrameToTimer3Captsel
enumerator kINPUTMUX_DcdcBurstActiveToTimer3Captsel
enumerator kINPUTMUX_Sai0TxSyncOutToTimer3Captsel
enumerator kINPUTMUX_Sai0RxSyncOutToTimer3Captsel
enumerator kINPUTMUX_Adc0IrqToTimer3Captsel
enumerator kINPUTMUX_Adc1IrqToTimer3Captsel

enumerator kINPUTMUX_Cmp0OutToTimer3Cptsel
enumerator kINPUTMUX_Cmp1OutToTimer3Cptsel
enumerator kINPUTMUX_Pwm0A0Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Pwm0A1Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Pwm0A2Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Pwm0A3Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Pwm1A0Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Pwm1A1Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Pwm1A2Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Pwm1A3Trig01ToTimer3Cptsel
enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer3Cptsel
enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut0AToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut0BToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut1AToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut1BToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut2AToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut2BToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut3AToTimer3Cptsel
enumerator kINPUTMUX_EvtgOut3BToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer3Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer3Cptsel

enumerator kINPUTMUX__Sai1TxSyncOutToTimer3Captsel
enumerator kINPUTMUX__Sai1RxSyncOutToTimer3Captsel
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__Usb0StartOfFrameToTimer4Captsel
enumerator kINPUTMUX__Usb1StartOfFrameToTimer4Captsel
enumerator kINPUTMUX__DcdcBurstActiveToTimer4Captsel
enumerator kINPUTMUX__Sai0TxSyncOutToTimer4Captsel
enumerator kINPUTMUX__Sai0RxSyncOutToTimer4Captsel
enumerator kINPUTMUX__Adc0IrqToTimer4Captsel
enumerator kINPUTMUX__Adc1IrqToTimer4Captsel
enumerator kINPUTMUX__Cmp0OutToTimer4Captsel
enumerator kINPUTMUX__Cmp1OutToTimer4Captsel
enumerator kINPUTMUX__Pwm0A0Trig01ToTimer4Captsel

enumerator kINPUTMUX_Pwm0A1Trig01ToTimer4Cptsel
enumerator kINPUTMUX_Pwm0A2Trig01ToTimer4Cptsel
enumerator kINPUTMUX_Pwm0A3Trig01ToTimer4Cptsel
enumerator kINPUTMUX_Pwm1A0Trig01ToTimer4Cptsel
enumerator kINPUTMUX_Pwm1A1Trig01ToTimer4Cptsel
enumerator kINPUTMUX_Pwm1A2Trig01ToTimer4Cptsel
enumerator kINPUTMUX_Pwm1A3Trig01ToTimer4Cptsel
enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer4Cptsel
enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut0AToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut0BToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut1AToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut1BToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut2AToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut2BToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut3AToTimer4Cptsel
enumerator kINPUTMUX_EvtgOut3BToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer4Cptsel
enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer4Cptsel
enumerator kINPUTMUX_Sai1TxSyncOutToTimer4Cptsel
enumerator kINPUTMUX_Sai1RxSyncOutToTimer4Cptsel
TIMER0 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_Usb0StartOfFrameToTimer0Trigger
enumerator kINPUTMUX_Usb1StartOfFrameToTimer0Trigger
enumerator kINPUTMUX_DcdcBurstActiveToTimer0Trigger
enumerator kINPUTMUX_Sai0TxSyncOutToTimer0Trigger
enumerator kINPUTMUX_Sai0RxSyncOutToTimer0Trigger
enumerator kINPUTMUX_Adc0IrqToTimer0Trigger
enumerator kINPUTMUX_Adc1IrqToTimer0Trigger
enumerator kINPUTMUX_Cmp0OutToTimer0Trigger
enumerator kINPUTMUX_Cmp1OutToTimer0Trigger
enumerator kINPUTMUX_Pwm0A0Trig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm0A1Trig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm0A2Trig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm0A3Trig01ToTimer0Trigger

enumerator kINPUTMUX_Pwm1A0Trig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm1A1Trig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm1A2Trig01ToTimer0Trigger
enumerator kINPUTMUX_Pwm1A3Trig01ToTimer0Trigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer0Trigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer0Trigger
enumerator kINPUTMUX_EvtgOut0AToTimer0Trigger
enumerator kINPUTMUX_EvtgOut0BToTimer0Trigger
enumerator kINPUTMUX_EvtgOut1AToTimer0Trigger
enumerator kINPUTMUX_EvtgOut1BToTimer0Trigger
enumerator kINPUTMUX_EvtgOut2AToTimer0Trigger
enumerator kINPUTMUX_EvtgOut2BToTimer0Trigger
enumerator kINPUTMUX_EvtgOut3AToTimer0Trigger
enumerator kINPUTMUX_EvtgOut3BToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer0Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer0Trigger
enumerator kINPUTMUX_Sai1TxSyncOutToTimer0Trigger
enumerator kINPUTMUX_Sai1RxSyncOutToTimer0Trigger
 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_Usb0StartOfFrameToTimer1Trigger
enumerator kINPUTMUX_Usb1StartOfFrameToTimer1Trigger
enumerator kINPUTMUX_DcdcBurstActiveToTimer1Trigger
enumerator kINPUTMUX_Sai0TxSyncOutToTimer1Trigger
enumerator kINPUTMUX_Sai0RxSyncOutToTimer1Trigger
enumerator kINPUTMUX_Adc0IrqToTimer1Trigger
enumerator kINPUTMUX_Adc1IrqToTimer1Trigger
enumerator kINPUTMUX_Cmp0OutToTimer1Trigger
enumerator kINPUTMUX_Cmp1OutToTimer1Trigger
enumerator kINPUTMUX_Pwm0A0Trig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm0A1Trig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm0A2Trig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm0A3Trig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm1A0Trig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm1A1Trig01ToTimer1Trigger
enumerator kINPUTMUX_Pwm1A2Trig01ToTimer1Trigger

enumerator kINPUTMUX_Pwm1A3Trig01ToTimer1Trigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer1Trigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer1Trigger
enumerator kINPUTMUX_EvtgOut0AToTimer1Trigger
enumerator kINPUTMUX_EvtgOut0BToTimer1Trigger
enumerator kINPUTMUX_EvtgOut1AToTimer1Trigger
enumerator kINPUTMUX_EvtgOut1BToTimer1Trigger
enumerator kINPUTMUX_EvtgOut2AToTimer1Trigger
enumerator kINPUTMUX_EvtgOut2BToTimer1Trigger
enumerator kINPUTMUX_EvtgOut3AToTimer1Trigger
enumerator kINPUTMUX_EvtgOut3BToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm0Trig0ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm0Trig1ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm0Trig2ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm1Trig0ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm1Trig1ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm1Trig2ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm2Trig0ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm2Trig1ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm2Trig2ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig0ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig1ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig2ToTimer1Trigger
enumerator kINPUTMUX_LpFlexcomm3Trig3ToTimer1Trigger
enumerator kINPUTMUX_Sai1TxSyncOutToTimer1Trigger
enumerator kINPUTMUX_Sai1RxSyncOutToTimer1Trigger
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_Usb0StartOfFrameToTimer2Trigger
enumerator kINPUTMUX_Usb1StartOfFrameToTimer2Trigger
enumerator kINPUTMUX_DcdcBurstActiveToTimer2Trigger
enumerator kINPUTMUX_Sai0TxSyncOutToTimer2Trigger
enumerator kINPUTMUX_Sai0RxSyncOutToTimer2Trigger
enumerator kINPUTMUX_Adc0IrqToTimer2Trigger
enumerator kINPUTMUX_Adc1IrqToTimer2Trigger
enumerator kINPUTMUX_Cmp0OutToTimer2Trigger
enumerator kINPUTMUX_Cmp1OutToTimer2Trigger
enumerator kINPUTMUX_Pwm0A0Trig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm0A1Trig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm0A2Trig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm0A3Trig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm1A0Trig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm1A1Trig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm1A2Trig01ToTimer2Trigger
enumerator kINPUTMUX_Pwm1A3Trig01ToTimer2Trigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer2Trigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer2Trigger

enumerator kINPUTMUX__EvtgOut0AToTimer2Trigger
enumerator kINPUTMUX__EvtgOut0BToTimer2Trigger
enumerator kINPUTMUX__EvtgOut1AToTimer2Trigger
enumerator kINPUTMUX__EvtgOut1BToTimer2Trigger
enumerator kINPUTMUX__EvtgOut2AToTimer2Trigger
enumerator kINPUTMUX__EvtgOut2BToTimer2Trigger
enumerator kINPUTMUX__EvtgOut3AToTimer2Trigger
enumerator kINPUTMUX__EvtgOut3BToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig0ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig1ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig2ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig0ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig1ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig2ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig0ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig1ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig2ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig0ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig1ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig2ToTimer2Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig3ToTimer2Trigger
enumerator kINPUTMUX__Sai1TxSyncOutToTimer2Trigger
enumerator kINPUTMUX__Sai1RxSyncOutToTimer2Trigger
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_Usb0StartOfFrameToTimer3Trigger
enumerator kINPUTMUX_Usb1StartOfFrameToTimer3Trigger
enumerator kINPUTMUX_DcdcBurstActiveToTimer3Trigger
enumerator kINPUTMUX_Sai0TxSyncOutToTimer3Trigger
enumerator kINPUTMUX_Sai0RxSyncOutToTimer3Trigger
enumerator kINPUTMUX_Adc0IrqToTimer3Trigger
enumerator kINPUTMUX_Adc1IrqToTimer3Trigger
enumerator kINPUTMUX_Cmp0OutToTimer3Trigger
enumerator kINPUTMUX_Cmp1OutToTimer3Trigger
enumerator kINPUTMUX_Pwm0A0Trig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm0A1Trig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm0A2Trig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm0A3Trig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1A0Trig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1A1Trig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1A2Trig01ToTimer3Trigger
enumerator kINPUTMUX_Pwm1A3Trig01ToTimer3Trigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToTimer3Trigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToTimer3Trigger
enumerator kINPUTMUX_EvtgOut0AToTimer3Trigger
enumerator kINPUTMUX_EvtgOut0BToTimer3Trigger
enumerator kINPUTMUX_EvtgOut1AToTimer3Trigger

enumerator kINPUTMUX__EvtgOut1BToTimer3Trigger
enumerator kINPUTMUX__EvtgOut2AToTimer3Trigger
enumerator kINPUTMUX__EvtgOut2BToTimer3Trigger
enumerator kINPUTMUX__EvtgOut3AToTimer3Trigger
enumerator kINPUTMUX__EvtgOut3BToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig0ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig1ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig2ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig0ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig1ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig2ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig0ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig1ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig2ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig0ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig1ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig2ToTimer3Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig3ToTimer3Trigger
enumerator kINPUTMUX__Sai1TxSyncOutToTimer3Trigger
enumerator kINPUTMUX__Sai1RxSyncOutToTimer3Trigger

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__Usb0StartOfFrameToTimer4Trigger
enumerator kINPUTMUX__Usb1StartOfFrameToTimer4Trigger
enumerator kINPUTMUX__DcdcBurstActiveToTimer4Trigger
enumerator kINPUTMUX__Sai0TxSyncOutToTimer4Trigger
enumerator kINPUTMUX__Sai0RxSyncOutToTimer4Trigger
enumerator kINPUTMUX__Adc0IrqToTimer4Trigger
enumerator kINPUTMUX__Adc1IrqToTimer4Trigger
enumerator kINPUTMUX__Cmp0OutToTimer4Trigger
enumerator kINPUTMUX__Cmp1OutToTimer4Trigger
enumerator kINPUTMUX__Pwm0A0Trig01ToTimer4Trigger
enumerator kINPUTMUX__Pwm0A1Trig01ToTimer4Trigger
enumerator kINPUTMUX__Pwm0A2Trig01ToTimer4Trigger
enumerator kINPUTMUX__Pwm0A3Trig01ToTimer4Trigger
enumerator kINPUTMUX__Pwm1A0Trig01ToTimer4Trigger
enumerator kINPUTMUX__Pwm1A1Trig01ToTimer4Trigger
enumerator kINPUTMUX__Pwm1A2Trig01ToTimer4Trigger
enumerator kINPUTMUX__Pwm1A3Trig01ToTimer4Trigger
enumerator kINPUTMUX__Qdc0CmpPosMatchToTimer4Trigger
enumerator kINPUTMUX__Qdc1CmpPosMatchToTimer4Trigger
enumerator kINPUTMUX__EvtgOut0AToTimer4Trigger
enumerator kINPUTMUX__EvtgOut0BToTimer4Trigger
enumerator kINPUTMUX__EvtgOut1AToTimer4Trigger
enumerator kINPUTMUX__EvtgOut1BToTimer4Trigger
enumerator kINPUTMUX__EvtgOut2AToTimer4Trigger
enumerator kINPUTMUX__EvtgOut2BToTimer4Trigger

enumerator kINPUTMUX__EvtgOut3AToTimer4Trigger
enumerator kINPUTMUX__EvtgOut3BToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig0ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig1ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm0Trig2ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig0ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig1ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm1Trig2ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig0ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig1ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm2Trig2ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig0ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig1ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig2ToTimer4Trigger
enumerator kINPUTMUX__LpFlexcomm3Trig3ToTimer4Trigger
enumerator kINPUTMUX__Sai1TxSyncOutToTimer4Trigger
enumerator kINPUTMUX__Sai1RxSyncOutToTimer4Trigger
 SMARTDMA arch B inputs.
enumerator kINPUTMUX__GpioPort0Pin0ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin1ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin2ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin3ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin4ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin5ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin6ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin7ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin12ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin13ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin14ToSmartDma
enumerator kINPUTMUX__GpioPort0Pin15ToSmartDma
enumerator kINPUTMUX__MrtCh0IrqToSmartDma
enumerator kINPUTMUX__MrtCh1IrqToSmartDma
enumerator kINPUTMUX__Ctimer4M3ToSmartDma

enumerator kINPUTMUX__Ctimer4M2ToSmartDma
enumerator kINPUTMUX__Ctimer3M3ToSmartDma
enumerator kINPUTMUX__Ctimer3M2ToSmartDma
enumerator kINPUTMUX__Ctimer1M3ToSmartDma
enumerator kINPUTMUX__Ctimer1M2ToSmartDma
enumerator kINPUTMUX__UtickIrqToSmartDma
enumerator kINPUTMUX__Wdt0IrqToSmartDma
enumerator kINPUTMUX__Adc0IrqToSmartDma
enumerator kINPUTMUX__Cmp0IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm7IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm6IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm5IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm4IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm3IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm2IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm1IrqToSmartDma
enumerator kINPUTMUX__LpFlexcomm0IrqToSmartDma
enumerator kINPUTMUX__Dma0IrqToSmartDma
enumerator kINPUTMUX__Dma1IrqToSmartDma
enumerator kINPUTMUX__SysIrqToSmartDma
enumerator kINPUTMUX__RtcComboIrqToSmartDma
enumerator kINPUTMUX__ArmTxevToSmartDma
enumerator kINPUTMUX__GpioIntBmatchToSmartDma
enumerator kINPUTMUX__Cmp0OutToSmartDma
enumerator kINPUTMUX__Usb0StartOfFrameIrqToSmartDma
enumerator kINPUTMUX__Usb1StartOfFrameIrqToSmartDma
enumerator kINPUTMUX__OsEventTimerIrqToSmartDma
enumerator kINPUTMUX__Adc1IrqToSmartDma
enumerator kINPUTMUX__Cmp01IrqToSmartDma
enumerator kINPUTMUX__Pwm0IrqToSmartDma
enumerator kINPUTMUX__Pwm1IrqToSmartDma
enumerator kINPUTMUX__Qdc0IrqToSmartDma
enumerator kINPUTMUX__Qdc1IrqToSmartDma

enumerator kINPUTMUX__EvtgOut0AToSmartDma
enumerator kINPUTMUX__EvtgOut1AToSmartDma
enumerator kINPUTMUX__Gpio1PinEventTrig0ToSmartDma
enumerator kINPUTMUX__Gpio1PinEventTrig1ToSmartDma
enumerator kINPUTMUX__Gpio2PinEventTrig0ToSmartDma
enumerator kINPUTMUX__Gpio2PinEventTrig1ToSmartDma
enumerator kINPUTMUX__Gpio3PinEventTrig0ToSmartDma
enumerator kINPUTMUX__Gpio3PinEventTrig1ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest0ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest1ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest2ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest3ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest4ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest5ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest6ToSmartDma
enumerator kINPUTMUX__FlexioShifterDmaRequest7ToSmartDma
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__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_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_GpioPort1Pin30ToPintsel

enumerator kINPUTMUX_GpioPort1Pin31ToPintsel

Selection for frequency measurement reference clock.

enumerator kINPUTMUX__ClkInToFreqmeasRef

enumerator kINPUTMUX__Fro12MToFreqmeasRef

enumerator kINPUTMUX__Fro144MToFreqmeasRef

enumerator kINPUTMUX__Osc32KToFreqmeasRef

enumerator kINPUTMUX__CpuAhbClkToFreqmeasRef

enumerator kINPUTMUX__FreqmeClkIn0ToFreqmeasRef

enumerator kINPUTMUX__FreqmeClkIn1ToFreqmeasRef

enumerator kINPUTMUX__EvtgOut0AToFreqmeasRef

enumerator kINPUTMUX__EvtgOut1AToFreqmeasRef

Selection for frequency measurement target clock.

enumerator kINPUTMUX__ClkInToFreqmeasTar

enumerator kINPUTMUX__Fro12MToFreqmeasTar

enumerator kINPUTMUX__Fro144MToFreqmeasTar

enumerator kINPUTMUX__Osc32KToFreqmeasTar

enumerator kINPUTMUX__CpuAhbClkToFreqmeasTar

enumerator kINPUTMUX__FreqmeClkIn0ToFreqmeasTar

enumerator kINPUTMUX__FreqmeClkIn1ToFreqmeasTar

enumerator kINPUTMUX__EvtgOut0AToFreqmeasTar

enumerator kINPUTMUX__EvtgOut1AToFreqmeasTar

Cmp0 Trigger.

enumerator kINPUTMUX__PinInt0ToCmp0Trigger

enumerator kINPUTMUX__PinInt6ToCmp0Trigger

enumerator kINPUTMUX__Ctimer0M3ToCmp0Trigger

enumerator kINPUTMUX__Ctimer1M3ToCmp0Trigger

enumerator kINPUTMUX__Ctimer2M3ToCmp0Trigger

enumerator kINPUTMUX__Ctimer0M0ToCmp0Trigger

enumerator kINPUTMUX__Ctimer4M0ToCmp0Trigger

enumerator kINPUTMUX__ArmTxevToCmp0Trigger

enumerator kINPUTMUX__GpioIntBmatchToCmp0Trigger

enumerator kINPUTMUX__Adc0Tcomp0ToCmp0Trigger

enumerator kINPUTMUX__Adc1Tcomp0ToCmp0Trigger

enumerator kINPUTMUX__Pwm0A0Trig01ToCmp0Trigger

enumerator kINPUTMUX__Pwm0A1Trig01ToCmp0Trigger

enumerator kINPUTMUX__Pwm0A2Trig01ToCmp0Trigger

enumerator kINPUTMUX_Pwm0A3Trig01ToCmp0Trigger
enumerator kINPUTMUX_Pwm1A0Trig01ToCmp0Trigger
enumerator kINPUTMUX_Pwm1A1Trig01ToCmp0Trigger
enumerator kINPUTMUX_Pwm1A2Trig01ToCmp0Trigger
enumerator kINPUTMUX_Pwm1A3Trig01ToCmp0Trigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToCmp0Trigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToCmp0Trigger
enumerator kINPUTMUX_EvtgOut0AToCmp0Trigger
enumerator kINPUTMUX_EvtgOut0BToCmp0Trigger
enumerator kINPUTMUX_EvtgOut1AToCmp0Trigger
enumerator kINPUTMUX_EvtgOut1BToCmp0Trigger
enumerator kINPUTMUX_EvtgOut2AToCmp0Trigger
enumerator kINPUTMUX_EvtgOut2BToCmp0Trigger
enumerator kINPUTMUX_EvtgOut3AToCmp0Trigger
enumerator kINPUTMUX_EvtgOut3BToCmp0Trigger
enumerator kINPUTMUX_Lptmr0ToCmp0Trigger
enumerator kINPUTMUX_Lptmr1ToCmp0Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig0ToCmp0Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig1ToCmp0Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig0ToCmp0Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig1ToCmp0Trigger
 Cmp1 Trigger.
enumerator kINPUTMUX_PinInt0ToCmp1Trigger
enumerator kINPUTMUX_PinInt7ToCmp1Trigger
enumerator kINPUTMUX_Ctimer0M3ToCmp1Trigger
enumerator kINPUTMUX_Ctimer1M3ToCmp1Trigger
enumerator kINPUTMUX_Ctimer2M3ToCmp1Trigger
enumerator kINPUTMUX_Ctimer3M1ToCmp1Trigger
enumerator kINPUTMUX_Ctimer4M1ToCmp1Trigger
enumerator kINPUTMUX_ArmTxevToCmp1Trigger
enumerator kINPUTMUX_GpioIntBmatchToCmp1Trigger
enumerator kINPUTMUX_Adc0Tcomp1ToCmp1Trigger
enumerator kINPUTMUX_Adc1Tcomp1ToCmp1Trigger

enumerator kINPUTMUX_Pwm0A0Trig01ToCmp1Trigger
enumerator kINPUTMUX_Pwm0A1Trig01ToCmp1Trigger
enumerator kINPUTMUX_Pwm0A2Trig01ToCmp1Trigger
enumerator kINPUTMUX_Pwm0A3Trig01ToCmp1Trigger
enumerator kINPUTMUX_Pwm1A0Trig01ToCmp1Trigger
enumerator kINPUTMUX_Pwm1A1Trig01ToCmp1Trigger
enumerator kINPUTMUX_Pwm1A2Trig01ToCmp1Trigger
enumerator kINPUTMUX_Pwm1A3Trig01ToCmp1Trigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToCmp1Trigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToCmp1Trigger
enumerator kINPUTMUX_EvtgOut0AToCmp1Trigger
enumerator kINPUTMUX_EvtgOut0BToCmp1Trigger
enumerator kINPUTMUX_EvtgOut1AToCmp1Trigger
enumerator kINPUTMUX_EvtgOut1BToCmp1Trigger
enumerator kINPUTMUX_EvtgOut2AToCmp1Trigger
enumerator kINPUTMUX_EvtgOut2BToCmp1Trigger
enumerator kINPUTMUX_EvtgOut3AToCmp1Trigger
enumerator kINPUTMUX_EvtgOut3BToCmp1Trigger
enumerator kINPUTMUX_Lptmr0ToCmp1Trigger
enumerator kINPUTMUX_Lptmr1ToCmp1Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig0ToCmp1Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig1ToCmp1Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig0ToCmp1Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig1ToCmp1Trigger
 Adc0 Trigger.
enumerator kINPUTMUX_PinInt0ToAdc0Trigger
enumerator kINPUTMUX_PinInt1ToAdc0Trigger
enumerator kINPUTMUX_Ctimer0M3ToAdc0Trigger
enumerator kINPUTMUX_Ctimer1M3ToAdc0Trigger
enumerator kINPUTMUX_Ctimer2M3ToAdc0Trigger
enumerator kINPUTMUX_Ctimer3M3ToAdc0Trigger
enumerator kINPUTMUX_Ctimer4M3ToAdc0Trigger
enumerator kINPUTMUX_DcdcBurstDoneTrigToAdc0Trigger

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__Cmp0OutToAdc0Trigger
enumerator kINPUTMUX__Cmp1OutToAdc0Trigger
enumerator kINPUTMUX__Pwm0A0Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0A0Trig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm0A1Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0A1Trig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm0A2Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0A2Trig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm0A3Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm0A3Trig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A0Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A0Trig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A1Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A1Trig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A2Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A2Trig1ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A3Trig0ToAdc0Trigger
enumerator kINPUTMUX__Pwm1A3Trig1ToAdc0Trigger
enumerator kINPUTMUX__Qdc0CmpPosMatchToAdc0Trigger
enumerator kINPUTMUX__Qdc1CmpPosMatchToAdc0Trigger
enumerator kINPUTMUX__EvtgOut0AToAdc0Trigger
enumerator kINPUTMUX__EvtgOut0BToAdc0Trigger
enumerator kINPUTMUX__EvtgOut1AToAdc0Trigger

enumerator kINPUTMUX__EvtgOut1BToAdc0Trigger
enumerator kINPUTMUX__EvtgOut2AToAdc0Trigger
enumerator kINPUTMUX__EvtgOut2BToAdc0Trigger
enumerator kINPUTMUX__EvtgOut3AToAdc0Trigger
enumerator kINPUTMUX__EvtgOut3BToAdc0Trigger
enumerator kINPUTMUX__Lptmr0ToAdc0Trigger
enumerator kINPUTMUX__Lptmr1ToAdc0Trigger
enumerator kINPUTMUX__FlexioCh0ToAdc0Trigger
enumerator kINPUTMUX__FlexioCh1ToAdc0Trigger
enumerator kINPUTMUX__FlexioCh2ToAdc0Trigger
enumerator kINPUTMUX__FlexioCh3ToAdc0Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig0ToAdc0Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig1ToAdc0Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig0ToAdc0Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig1ToAdc0Trigger
enumerator kINPUTMUX__WuuToAdc0Trigger
 Adc1 Trigger.
enumerator kINPUTMUX__PinInt0ToAdc1Trigger
enumerator kINPUTMUX__PinInt2ToAdc1Trigger
enumerator kINPUTMUX__Ctimer0M3ToAdc1Trigger
enumerator kINPUTMUX__Ctimer1M3ToAdc1Trigger
enumerator kINPUTMUX__Ctimer2M3ToAdc1Trigger
enumerator kINPUTMUX__Ctimer3M2ToAdc1Trigger
enumerator kINPUTMUX__Ctimer4M1ToAdc1Trigger
enumerator kINPUTMUX__DcdcBurstDoneTrigToAdc1Trigger
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__Cmp0OutToAdc1Trigger
enumerator kINPUTMUX__Cmp1OutToAdc1Trigger
enumerator kINPUTMUX__Pwm0A0Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm0A0Trig1ToAdc1Trigger
enumerator kINPUTMUX__Pwm0A1Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm0A1Trig1ToAdc1Trigger
enumerator kINPUTMUX__Pwm0A2Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm0A2Trig1ToAdc1Trigger
enumerator kINPUTMUX__Pwm0A3Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm0A3Trig1ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A0Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A0Trig1ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A1Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A1Trig1ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A2Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A2Trig1ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A3Trig0ToAdc1Trigger
enumerator kINPUTMUX__Pwm1A3Trig1ToAdc1Trigger
enumerator kINPUTMUX__Qdc0CmpPosMatchToAdc1Trigger
enumerator kINPUTMUX__Qdc1CmpPosMatchToAdc1Trigger
enumerator kINPUTMUX__EvtgOut0AToAdc1Trigger
enumerator kINPUTMUX__EvtgOut0BToAdc1Trigger
enumerator kINPUTMUX__EvtgOut1AToAdc1Trigger
enumerator kINPUTMUX__EvtgOut1BToAdc1Trigger
enumerator kINPUTMUX__EvtgOut2AToAdc1Trigger
enumerator kINPUTMUX__EvtgOut2BToAdc1Trigger
enumerator kINPUTMUX__EvtgOut3AToAdc1Trigger
enumerator kINPUTMUX__EvtgOut3BToAdc1Trigger
enumerator kINPUTMUX__Lptmr0ToAdc1Trigger
enumerator kINPUTMUX__Lptmr1ToAdc1Trigger
enumerator kINPUTMUX__FlexioCh0ToAdc1Trigger

enumerator kINPUTMUX_FlexioCh1ToAdc1Trigger
enumerator kINPUTMUX_FlexioCh2ToAdc1Trigger
enumerator kINPUTMUX_FlexioCh3ToAdc1Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig0ToAdc1Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig1ToAdc1Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig0ToAdc1Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig1ToAdc1Trigger
enumerator kINPUTMUX_WuuToAdc1Trigger

QDC0 Trigger Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc0Trigger
enumerator kINPUTMUX_PinInt4ToQdc0Trigger
enumerator kINPUTMUX_Ctimer0M3ToQdc0Trigger
enumerator kINPUTMUX_Ctimer1M3ToQdc0Trigger
enumerator kINPUTMUX_Ctimer2M3ToQdc0Trigger
enumerator kINPUTMUX_Ctimer1M0ToQdc0Trigger
enumerator kINPUTMUX_Ctimer3M0ToQdc0Trigger
enumerator kINPUTMUX_ArmTxevToQdc0Trigger
enumerator kINPUTMUX_GpioIntBmatchToQdc0Trigger
enumerator kINPUTMUX_Adc0Tcomp0ToQdc0Trigger
enumerator kINPUTMUX_Adc0Tcomp1ToQdc0Trigger
enumerator kINPUTMUX_Adc0Tcomp2ToQdc0Trigger
enumerator kINPUTMUX_Adc0Tcomp3ToQdc0Trigger
enumerator kINPUTMUX_Adc1Tcomp0ToQdc0Trigger
enumerator kINPUTMUX_Adc1Tcomp1ToQdc0Trigger
enumerator kINPUTMUX_Adc1Tcomp2ToQdc0Trigger
enumerator kINPUTMUX_Adc1Tcomp3ToQdc0Trigger
enumerator kINPUTMUX_Cmp0OutToQdc0Trigger
enumerator kINPUTMUX_Cmp1OutToQdc0Trigger
enumerator kINPUTMUX_Pwm1A0Trig0ToQdc0Trigger
enumerator kINPUTMUX_Pwm1A0Trig1ToQdc0Trigger
enumerator kINPUTMUX_Pwm1A1Trig0ToQdc0Trigger
enumerator kINPUTMUX_Pwm1A1Trig1ToQdc0Trigger
enumerator kINPUTMUX_Pwm1A2Trig0ToQdc0Trigger

enumerator kINPUTMUX_Pwm1A2Trig1ToQdc0Trigger
enumerator kINPUTMUX_Pwm1A3Trig0ToQdc0Trigger
enumerator kINPUTMUX_Pwm1A3Trig1ToQdc0Trigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc0Trigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc0Trigger
enumerator kINPUTMUX_EvtgOut0AtoQdc0Trigger
enumerator kINPUTMUX_EvtgOut0BtoQdc0Trigger
enumerator kINPUTMUX_EvtgOut1AtoQdc0Trigger
enumerator kINPUTMUX_EvtgOut1BtoQdc0Trigger
enumerator kINPUTMUX_EvtgOut2AtoQdc0Trigger
enumerator kINPUTMUX_EvtgOut2BtoQdc0Trigger
enumerator kINPUTMUX_EvtgOut3AtoQdc0Trigger
enumerator kINPUTMUX_EvtgOut3BtoQdc0Trigger
enumerator kINPUTMUX_TrigIn0ToQdc0Trigger
enumerator kINPUTMUX_TrigIn1ToQdc0Trigger
enumerator kINPUTMUX_TrigIn2ToQdc0Trigger
enumerator kINPUTMUX_TrigIn3ToQdc0Trigger
enumerator kINPUTMUX_TrigIn4ToQdc0Trigger
enumerator kINPUTMUX_TrigIn5ToQdc0Trigger
enumerator kINPUTMUX_TrigIn6ToQdc0Trigger
enumerator kINPUTMUX_TrigIn7ToQdc0Trigger
enumerator kINPUTMUX_TrigIn8ToQdc0Trigger
enumerator kINPUTMUX_TrigIn9ToQdc0Trigger
QDC0 Home Input Connections.
enumerator kINPUTMUX_PinInt0ToQdc0Home
enumerator kINPUTMUX_PinInt4ToQdc0Home
enumerator kINPUTMUX_Ctimer0M3ToQdc0Home
enumerator kINPUTMUX_Ctimer1M3ToQdc0Home
enumerator kINPUTMUX_Ctimer2M3ToQdc0Home
enumerator kINPUTMUX_Ctimer1M0ToQdc0Home
enumerator kINPUTMUX_Ctimer3M0ToQdc0Home
enumerator kINPUTMUX_ArmTxevToQdc0Home
enumerator kINPUTMUX_GpioIntBmatchToQdc0Home

enumerator kINPUTMUX__Adc0Tcomp0ToQdc0Home
enumerator kINPUTMUX__Adc0Tcomp1ToQdc0Home
enumerator kINPUTMUX__Adc0Tcomp2ToQdc0Home
enumerator kINPUTMUX__Adc0Tcomp3ToQdc0Home
enumerator kINPUTMUX__Adc1Tcomp0ToQdc0Home
enumerator kINPUTMUX__Adc1Tcomp1ToQdc0Home
enumerator kINPUTMUX__Adc1Tcomp2ToQdc0Home
enumerator kINPUTMUX__Adc1Tcomp3ToQdc0Home
enumerator kINPUTMUX__Cmp0OutToQdc0Home
enumerator kINPUTMUX__Cmp1OutToQdc0Home
enumerator kINPUTMUX__Pwm1A0Trig0ToQdc0Home
enumerator kINPUTMUX__Pwm1A0Trig1ToQdc0Home
enumerator kINPUTMUX__Pwm1A1Trig0ToQdc0Home
enumerator kINPUTMUX__Pwm1A1Trig1ToQdc0Home
enumerator kINPUTMUX__Pwm1A2Trig0ToQdc0Home
enumerator kINPUTMUX__Pwm1A2Trig1ToQdc0Home
enumerator kINPUTMUX__Pwm1A3Trig0ToQdc0Home
enumerator kINPUTMUX__Pwm1A3Trig1ToQdc0Home
enumerator kINPUTMUX__Qdc0CmpPosMatchToQdc0Home
enumerator kINPUTMUX__Qdc1CmpPosMatchToQdc0Home
enumerator kINPUTMUX__EvtgOut0AToQdc0Home
enumerator kINPUTMUX__EvtgOut0BToQdc0Home
enumerator kINPUTMUX__EvtgOut1AToQdc0Home
enumerator kINPUTMUX__EvtgOut1BToQdc0Home
enumerator kINPUTMUX__EvtgOut2AToQdc0Home
enumerator kINPUTMUX__EvtgOut2BToQdc0Home
enumerator kINPUTMUX__EvtgOut3AToQdc0Home
enumerator kINPUTMUX__EvtgOut3BToQdc0Home
enumerator kINPUTMUX__TrigIn0ToQdc0Home
enumerator kINPUTMUX__TrigIn1ToQdc0Home
enumerator kINPUTMUX__TrigIn2ToQdc0Home
enumerator kINPUTMUX__TrigIn3ToQdc0Home
enumerator kINPUTMUX__TrigIn4ToQdc0Home

enumerator kINPUTMUX__TrigIn5ToQdc0Home

enumerator kINPUTMUX__TrigIn6ToQdc0Home

enumerator kINPUTMUX__TrigIn7ToQdc0Home

enumerator kINPUTMUX__TrigIn8ToQdc0Home

enumerator kINPUTMUX__TrigIn9ToQdc0Home

QDC0 Index Input Connections.

enumerator kINPUTMUX__PinInt0ToQdc0Index

enumerator kINPUTMUX__PinInt4ToQdc0Index

enumerator kINPUTMUX__Ctimer0M3ToQdc0Index

enumerator kINPUTMUX__Ctimer1M3ToQdc0Index

enumerator kINPUTMUX__Ctimer2M3ToQdc0Index

enumerator kINPUTMUX__Ctimer1M0ToQdc0Index

enumerator kINPUTMUX__Ctimer3M0ToQdc0Index

enumerator kINPUTMUX__ArmTxevToQdc0Index

enumerator kINPUTMUX__GpioIntBmatchToQdc0Index

enumerator kINPUTMUX__Adc0Tcomp0ToQdc0Index

enumerator kINPUTMUX__Adc0Tcomp1ToQdc0Index

enumerator kINPUTMUX__Adc0Tcomp2ToQdc0Index

enumerator kINPUTMUX__Adc0Tcomp3ToQdc0Index

enumerator kINPUTMUX__Adc1Tcomp0ToQdc0Index

enumerator kINPUTMUX__Adc1Tcomp1ToQdc0Index

enumerator kINPUTMUX__Adc1Tcomp2ToQdc0Index

enumerator kINPUTMUX__Adc1Tcomp3ToQdc0Index

enumerator kINPUTMUX__Cmp0OutToQdc0Index

enumerator kINPUTMUX__Cmp1OutToQdc0Index

enumerator kINPUTMUX__Pwm1A0Trig0ToQdc0Index

enumerator kINPUTMUX__Pwm1A0Trig1ToQdc0Index

enumerator kINPUTMUX__Pwm1A1Trig0ToQdc0Index

enumerator kINPUTMUX__Pwm1A1Trig1ToQdc0Index

enumerator kINPUTMUX__Pwm1A2Trig0ToQdc0Index

enumerator kINPUTMUX__Pwm1A2Trig1ToQdc0Index

enumerator kINPUTMUX__Pwm1A3Trig0ToQdc0Index

enumerator kINPUTMUX__Pwm1A3Trig1ToQdc0Index

enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc0Index
enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc0Index
enumerator kINPUTMUX_EvtgOut0AToQdc0Index
enumerator kINPUTMUX_EvtgOut0BToQdc0Index
enumerator kINPUTMUX_EvtgOut1AToQdc0Index
enumerator kINPUTMUX_EvtgOut1BToQdc0Index
enumerator kINPUTMUX_EvtgOut2AToQdc0Index
enumerator kINPUTMUX_EvtgOut2BToQdc0Index
enumerator kINPUTMUX_EvtgOut3AToQdc0Index
enumerator kINPUTMUX_EvtgOut3BToQdc0Index
enumerator kINPUTMUX_TrigIn0ToQdc0Index
enumerator kINPUTMUX_TrigIn1ToQdc0Index
enumerator kINPUTMUX_TrigIn2ToQdc0Index
enumerator kINPUTMUX_TrigIn3ToQdc0Index
enumerator kINPUTMUX_TrigIn4ToQdc0Index
enumerator kINPUTMUX_TrigIn5ToQdc0Index
enumerator kINPUTMUX_TrigIn6ToQdc0Index
enumerator kINPUTMUX_TrigIn7ToQdc0Index
enumerator kINPUTMUX_TrigIn8ToQdc0Index
enumerator kINPUTMUX_TrigIn9ToQdc0Index

QDC0 Phaseb Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc0Phaseb
enumerator kINPUTMUX_PinInt4ToQdc0Phaseb
enumerator kINPUTMUX_Ctimer0M3ToQdc0Phaseb
enumerator kINPUTMUX_Ctimer1M3ToQdc0Phaseb
enumerator kINPUTMUX_Ctimer2M3ToQdc0Phaseb
enumerator kINPUTMUX_Ctimer1M0ToQdc0Phaseb
enumerator kINPUTMUX_Ctimer3M0ToQdc0Phaseb
enumerator kINPUTMUX_ArmTxevToQdc0Phaseb
enumerator kINPUTMUX_GpioIntBmatchToQdc0Phaseb
enumerator kINPUTMUX_Adc0Tcomp0ToQdc0Phaseb
enumerator kINPUTMUX_Adc0Tcomp1ToQdc0Phaseb
enumerator kINPUTMUX_Adc0Tcomp2ToQdc0Phaseb

enumerator kINPUTMUX__Adc0Tcomp3ToQdc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp0ToQdc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp1ToQdc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp2ToQdc0Phaseb
enumerator kINPUTMUX__Adc1Tcomp3ToQdc0Phaseb
enumerator kINPUTMUX__Cmp0OutToQdc0Phaseb
enumerator kINPUTMUX__Cmp1OutToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A0Trig0ToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A0Trig1ToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A1Trig0ToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A1Trig1ToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A2Trig0ToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A2Trig1ToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A3Trig0ToQdc0Phaseb
enumerator kINPUTMUX__Pwm1A3Trig1ToQdc0Phaseb
enumerator kINPUTMUX__Qdc0CmpPosMatchToQdc0Phaseb
enumerator kINPUTMUX__Qdc1CmpPosMatchToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut0AToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut0BToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut1AToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut1BToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut2AToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut2BToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut3AToQdc0Phaseb
enumerator kINPUTMUX__EvtgOut3BToQdc0Phaseb
enumerator kINPUTMUX__TrigIn0ToQdc0Phaseb
enumerator kINPUTMUX__TrigIn1ToQdc0Phaseb
enumerator kINPUTMUX__TrigIn2ToQdc0Phaseb
enumerator kINPUTMUX__TrigIn3ToQdc0Phaseb
enumerator kINPUTMUX__TrigIn4ToQdc0Phaseb
enumerator kINPUTMUX__TrigIn5ToQdc0Phaseb
enumerator kINPUTMUX__TrigIn6ToQdc0Phaseb
enumerator kINPUTMUX__TrigIn7ToQdc0Phaseb

enumerator kINPUTMUX__TrigIn8ToQdc0Phaseb

enumerator kINPUTMUX__TrigIn9ToQdc0Phaseb

QDC0 Phasea Input Connections.

enumerator kINPUTMUX__PinInt0ToQdc0Phasea

enumerator kINPUTMUX__PinInt4ToQdc0Phasea

enumerator kINPUTMUX__Ctimer0M3ToQdc0Phasea

enumerator kINPUTMUX__Ctimer1M3ToQdc0Phasea

enumerator kINPUTMUX__Ctimer2M3ToQdc0Phasea

enumerator kINPUTMUX__Ctimer1M0ToQdc0Phasea

enumerator kINPUTMUX__Ctimer3M0ToQdc0Phasea

enumerator kINPUTMUX__ArmTxevToQdc0Phasea

enumerator kINPUTMUX__GpioIntBmatchToQdc0Phasea

enumerator kINPUTMUX__Adc0Tcomp0ToQdc0Phasea

enumerator kINPUTMUX__Adc0Tcomp1ToQdc0Phasea

enumerator kINPUTMUX__Adc0Tcomp2ToQdc0Phasea

enumerator kINPUTMUX__Adc0Tcomp3ToQdc0Phasea

enumerator kINPUTMUX__Adc1Tcomp0ToQdc0Phasea

enumerator kINPUTMUX__Adc1Tcomp1ToQdc0Phasea

enumerator kINPUTMUX__Adc1Tcomp2ToQdc0Phasea

enumerator kINPUTMUX__Adc1Tcomp3ToQdc0Phasea

enumerator kINPUTMUX__Cmp0OutToQdc0Phasea

enumerator kINPUTMUX__Cmp1OutToQdc0Phasea

enumerator kINPUTMUX__Pwm1A0Trig0ToQdc0Phasea

enumerator kINPUTMUX__Pwm1A0Trig1ToQdc0Phasea

enumerator kINPUTMUX__Pwm1A1Trig0ToQdc0Phasea

enumerator kINPUTMUX__Pwm1A1Trig1ToQdc0Phasea

enumerator kINPUTMUX__Pwm1A2Trig0ToQdc0Phasea

enumerator kINPUTMUX__Pwm1A2Trig1ToQdc0Phasea

enumerator kINPUTMUX__Pwm1A3Trig0ToQdc0Phasea

enumerator kINPUTMUX__Pwm1A3Trig1ToQdc0Phasea

enumerator kINPUTMUX__Qdc0CmpPosMatchToQdc0Phasea

enumerator kINPUTMUX__Qdc1CmpPosMatchToQdc0Phasea

enumerator kINPUTMUX__EvtgOut0AtoQdc0Phasea

enumerator kINPUTMUX__EvtgOut0BToQdc0Phasea

enumerator kINPUTMUX__EvtgOut1AToQdc0Phasea

enumerator kINPUTMUX__EvtgOut1BToQdc0Phasea

enumerator kINPUTMUX__EvtgOut2AToQdc0Phasea

enumerator kINPUTMUX__EvtgOut2BToQdc0Phasea

enumerator kINPUTMUX__EvtgOut3AToQdc0Phasea

enumerator kINPUTMUX__EvtgOut3BToQdc0Phasea

enumerator kINPUTMUX__TrigIn0ToQdc0Phasea

enumerator kINPUTMUX__TrigIn1ToQdc0Phasea

enumerator kINPUTMUX__TrigIn2ToQdc0Phasea

enumerator kINPUTMUX__TrigIn3ToQdc0Phasea

enumerator kINPUTMUX__TrigIn4ToQdc0Phasea

enumerator kINPUTMUX__TrigIn5ToQdc0Phasea

enumerator kINPUTMUX__TrigIn6ToQdc0Phasea

enumerator kINPUTMUX__TrigIn7ToQdc0Phasea

enumerator kINPUTMUX__TrigIn8ToQdc0Phasea

enumerator kINPUTMUX__TrigIn9ToQdc0Phasea

QDC1 Trigger Input Connections.

enumerator kINPUTMUX__PinInt0ToQdc1Trigger

enumerator kINPUTMUX__PinInt4ToQdc1Trigger

enumerator kINPUTMUX__Ctimer0M3ToQdc1Trigger

enumerator kINPUTMUX__Ctimer1M3ToQdc1Trigger

enumerator kINPUTMUX__Ctimer2M3ToQdc1Trigger

enumerator kINPUTMUX__Ctimer1M0ToQdc1Trigger

enumerator kINPUTMUX__Ctimer3M0ToQdc1Trigger

enumerator kINPUTMUX__ArmTxevToQdc1Trigger

enumerator kINPUTMUX__GpioIntBmatchToQdc1Trigger

enumerator kINPUTMUX__Adc0Tcomp0ToQdc1Trigger

enumerator kINPUTMUX__Adc0Tcomp1ToQdc1Trigger

enumerator kINPUTMUX__Adc0Tcomp2ToQdc1Trigger

enumerator kINPUTMUX__Adc0Tcomp3ToQdc1Trigger

enumerator kINPUTMUX__Adc1Tcomp0ToQdc1Trigger

enumerator kINPUTMUX__Adc1Tcomp1ToQdc1Trigger

enumerator kINPUTMUX__Adc1Tcomp2ToQdc1Trigger
enumerator kINPUTMUX__Adc1Tcomp3ToQdc1Trigger
enumerator kINPUTMUX__Cmp0OutToQdc1Trigger
enumerator kINPUTMUX__Cmp1OutToQdc1Trigger
enumerator kINPUTMUX__Pwm1A0Trig0ToQdc1Trigger
enumerator kINPUTMUX__Pwm1A0Trig1ToQdc1Trigger
enumerator kINPUTMUX__Pwm1A1Trig0ToQdc1Trigger
enumerator kINPUTMUX__Pwm1A1Trig1ToQdc1Trigger
enumerator kINPUTMUX__Pwm1A2Trig0ToQdc1Trigger
enumerator kINPUTMUX__Pwm1A2Trig1ToQdc1Trigger
enumerator kINPUTMUX__Pwm1A3Trig0ToQdc1Trigger
enumerator kINPUTMUX__Pwm1A3Trig1ToQdc1Trigger
enumerator kINPUTMUX__Qdc0CmpPosMatchToQdc1Trigger
enumerator kINPUTMUX__Qdc1CmpPosMatchToQdc1Trigger
enumerator kINPUTMUX__EvtgOut0AToQdc1Trigger
enumerator kINPUTMUX__EvtgOut0BToQdc1Trigger
enumerator kINPUTMUX__EvtgOut1AToQdc1Trigger
enumerator kINPUTMUX__EvtgOut1BToQdc1Trigger
enumerator kINPUTMUX__EvtgOut2AToQdc1Trigger
enumerator kINPUTMUX__EvtgOut2BToQdc1Trigger
enumerator kINPUTMUX__EvtgOut3AToQdc1Trigger
enumerator kINPUTMUX__EvtgOut3BToQdc1Trigger
enumerator kINPUTMUX__TrigIn0ToQdc1Trigger
enumerator kINPUTMUX__TrigIn1ToQdc1Trigger
enumerator kINPUTMUX__TrigIn2ToQdc1Trigger
enumerator kINPUTMUX__TrigIn3ToQdc1Trigger
enumerator kINPUTMUX__TrigIn4ToQdc1Trigger
enumerator kINPUTMUX__TrigIn5ToQdc1Trigger
enumerator kINPUTMUX__TrigIn6ToQdc1Trigger
enumerator kINPUTMUX__TrigIn7ToQdc1Trigger
enumerator kINPUTMUX__TrigIn8ToQdc1Trigger
enumerator kINPUTMUX__TrigIn9ToQdc1Trigger

QDC1 Home Input Connections.

enumerator kINPUTMUX_PinInt0ToQdc1Home
enumerator kINPUTMUX_PinInt4ToQdc1Home
enumerator kINPUTMUX_Ctimer0M3ToQdc1Home
enumerator kINPUTMUX_Ctimer1M3ToQdc1Home
enumerator kINPUTMUX_Ctimer2M3ToQdc1Home
enumerator kINPUTMUX_Ctimer1M0ToQdc1Home
enumerator kINPUTMUX_Ctimer3M0ToQdc1Home
enumerator kINPUTMUX_ArmTxevToQdc1Home
enumerator kINPUTMUX_GpioIntBmatchToQdc1Home
enumerator kINPUTMUX_Adc0Tcomp0ToQdc1Home
enumerator kINPUTMUX_Adc0Tcomp1ToQdc1Home
enumerator kINPUTMUX_Adc0Tcomp2ToQdc1Home
enumerator kINPUTMUX_Adc0Tcomp3ToQdc1Home
enumerator kINPUTMUX_Adc1Tcomp0ToQdc1Home
enumerator kINPUTMUX_Adc1Tcomp1ToQdc1Home
enumerator kINPUTMUX_Adc1Tcomp2ToQdc1Home
enumerator kINPUTMUX_Adc1Tcomp3ToQdc1Home
enumerator kINPUTMUX_Cmp0OutToQdc1Home
enumerator kINPUTMUX_Cmp1OutToQdc1Home
enumerator kINPUTMUX_Pwm1A0Trig0ToQdc1Home
enumerator kINPUTMUX_Pwm1A0Trig1ToQdc1Home
enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Home
enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Home
enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Home
enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Home
enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Home
enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Home
enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Home
enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Home
enumerator kINPUTMUX_EvtgOut0AToQdc1Home
enumerator kINPUTMUX_EvtgOut0BToQdc1Home
enumerator kINPUTMUX_EvtgOut1AToQdc1Home
enumerator kINPUTMUX_EvtgOut1BToQdc1Home

enumerator kINPUTMUX__EvtgOut2AToQdc1Home

enumerator kINPUTMUX__EvtgOut2BToQdc1Home

enumerator kINPUTMUX__EvtgOut3AToQdc1Home

enumerator kINPUTMUX__EvtgOut3BToQdc1Home

enumerator kINPUTMUX__TrigIn0ToQdc1Home

enumerator kINPUTMUX__TrigIn1ToQdc1Home

enumerator kINPUTMUX__TrigIn2ToQdc1Home

enumerator kINPUTMUX__TrigIn3ToQdc1Home

enumerator kINPUTMUX__TrigIn4ToQdc1Home

enumerator kINPUTMUX__TrigIn5ToQdc1Home

enumerator kINPUTMUX__TrigIn6ToQdc1Home

enumerator kINPUTMUX__TrigIn7ToQdc1Home

enumerator kINPUTMUX__TrigIn8ToQdc1Home

enumerator kINPUTMUX__TrigIn9ToQdc1Home

QDC1 Index Input Connections.

enumerator kINPUTMUX__PinInt0ToQdc1Index

enumerator kINPUTMUX__PinInt4ToQdc1Index

enumerator kINPUTMUX__Ctimer0M3ToQdc1Index

enumerator kINPUTMUX__Ctimer1M3ToQdc1Index

enumerator kINPUTMUX__Ctimer2M3ToQdc1Index

enumerator kINPUTMUX__Ctimer1M0ToQdc1Index

enumerator kINPUTMUX__Ctimer3M0ToQdc1Index

enumerator kINPUTMUX__ArmTxevToQdc1Index

enumerator kINPUTMUX__GpioIntBmatchToQdc1Index

enumerator kINPUTMUX__Adc0Tcomp0ToQdc1Index

enumerator kINPUTMUX__Adc0Tcomp1ToQdc1Index

enumerator kINPUTMUX__Adc0Tcomp2ToQdc1Index

enumerator kINPUTMUX__Adc0Tcomp3ToQdc1Index

enumerator kINPUTMUX__Adc1Tcomp0ToQdc1Index

enumerator kINPUTMUX__Adc1Tcomp1ToQdc1Index

enumerator kINPUTMUX__Adc1Tcomp2ToQdc1Index

enumerator kINPUTMUX__Adc1Tcomp3ToQdc1Index

enumerator kINPUTMUX__Cmp0OutToQdc1Index

enumerator kINPUTMUX_Cmp1OutToQdc1Index
enumerator kINPUTMUX_Pwm1A0Trig0ToQdc1Index
enumerator kINPUTMUX_Pwm1A0Trig1ToQdc1Index
enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Index
enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Index
enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Index
enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Index
enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Index
enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Index
enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Index
enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Index
enumerator kINPUTMUX_EvtgOut0AToQdc1Index
enumerator kINPUTMUX_EvtgOut0BToQdc1Index
enumerator kINPUTMUX_EvtgOut1AToQdc1Index
enumerator kINPUTMUX_EvtgOut1BToQdc1Index
enumerator kINPUTMUX_EvtgOut2AToQdc1Index
enumerator kINPUTMUX_EvtgOut2BToQdc1Index
enumerator kINPUTMUX_EvtgOut3AToQdc1Index
enumerator kINPUTMUX_EvtgOut3BToQdc1Index
enumerator kINPUTMUX_TrigIn0ToQdc1Index
enumerator kINPUTMUX_TrigIn1ToQdc1Index
enumerator kINPUTMUX_TrigIn2ToQdc1Index
enumerator kINPUTMUX_TrigIn3ToQdc1Index
enumerator kINPUTMUX_TrigIn4ToQdc1Index
enumerator kINPUTMUX_TrigIn5ToQdc1Index
enumerator kINPUTMUX_TrigIn6ToQdc1Index
enumerator kINPUTMUX_TrigIn7ToQdc1Index
enumerator kINPUTMUX_TrigIn8ToQdc1Index
enumerator kINPUTMUX_TrigIn9ToQdc1Index
QDC1 Phaseb Input Connections.
enumerator kINPUTMUX_PinInt0ToQdc1Phaseb
enumerator kINPUTMUX_PinInt4ToQdc1Phaseb
enumerator kINPUTMUX_Ctimer0M3ToQdc1Phaseb

enumerator kINPUTMUX__Ctimer1M3ToQdc1Phaseb
enumerator kINPUTMUX__Ctimer2M3ToQdc1Phaseb
enumerator kINPUTMUX__Ctimer1M0ToQdc1Phaseb
enumerator kINPUTMUX__Ctimer3M0ToQdc1Phaseb
enumerator kINPUTMUX__ArmTxevToQdc1Phaseb
enumerator kINPUTMUX__GpioIntBmatchToQdc1Phaseb
enumerator kINPUTMUX__Adc0Tcomp0ToQdc1Phaseb
enumerator kINPUTMUX__Adc0Tcomp1ToQdc1Phaseb
enumerator kINPUTMUX__Adc0Tcomp2ToQdc1Phaseb
enumerator kINPUTMUX__Adc0Tcomp3ToQdc1Phaseb
enumerator kINPUTMUX__Adc1Tcomp0ToQdc1Phaseb
enumerator kINPUTMUX__Adc1Tcomp1ToQdc1Phaseb
enumerator kINPUTMUX__Adc1Tcomp2ToQdc1Phaseb
enumerator kINPUTMUX__Adc1Tcomp3ToQdc1Phaseb
enumerator kINPUTMUX__Cmp0OutToQdc1Phaseb
enumerator kINPUTMUX__Cmp1OutToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A0Trig0ToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A0Trig1ToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A1Trig0ToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A1Trig1ToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A2Trig0ToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A2Trig1ToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A3Trig0ToQdc1Phaseb
enumerator kINPUTMUX__Pwm1A3Trig1ToQdc1Phaseb
enumerator kINPUTMUX__Qdc0CmpPosMatchToQdc1Phaseb
enumerator kINPUTMUX__Qdc1CmpPosMatchToQdc1Phaseb
enumerator kINPUTMUX__EvtgOut0AToQdc1Phaseb
enumerator kINPUTMUX__EvtgOut0BToQdc1Phaseb
enumerator kINPUTMUX__EvtgOut1AToQdc1Phaseb
enumerator kINPUTMUX__EvtgOut1BToQdc1Phaseb
enumerator kINPUTMUX__EvtgOut2AToQdc1Phaseb
enumerator kINPUTMUX__EvtgOut2BToQdc1Phaseb
enumerator kINPUTMUX__EvtgOut3AToQdc1Phaseb

enumerator kINPUTMUX__EvtgOut3BToQdc1Phaseb

enumerator kINPUTMUX__TrigIn0ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn1ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn2ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn3ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn4ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn5ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn6ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn7ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn8ToQdc1Phaseb

enumerator kINPUTMUX__TrigIn9ToQdc1Phaseb

QDC1 Phasea Input Connections.

enumerator kINPUTMUX__PinInt0ToQdc1Phasea

enumerator kINPUTMUX__PinInt4ToQdc1Phasea

enumerator kINPUTMUX__Ctimer0M3ToQdc1Phasea

enumerator kINPUTMUX__Ctimer1M3ToQdc1Phasea

enumerator kINPUTMUX__Ctimer2M3ToQdc1Phasea

enumerator kINPUTMUX__Ctimer1M0ToQdc1Phasea

enumerator kINPUTMUX__Ctimer3M0ToQdc1Phasea

enumerator kINPUTMUX__ArmTxevToQdc1Phasea

enumerator kINPUTMUX__GpioIntBmatchToQdc1Phasea

enumerator kINPUTMUX__Adc0Tcomp0ToQdc1Phasea

enumerator kINPUTMUX__Adc0Tcomp1ToQdc1Phasea

enumerator kINPUTMUX__Adc0Tcomp2ToQdc1Phasea

enumerator kINPUTMUX__Adc0Tcomp3ToQdc1Phasea

enumerator kINPUTMUX__Adc1Tcomp0ToQdc1Phasea

enumerator kINPUTMUX__Adc1Tcomp1ToQdc1Phasea

enumerator kINPUTMUX__Adc1Tcomp2ToQdc1Phasea

enumerator kINPUTMUX__Adc1Tcomp3ToQdc1Phasea

enumerator kINPUTMUX__Cmp0OutToQdc1Phasea

enumerator kINPUTMUX__Cmp1OutToQdc1Phasea

enumerator kINPUTMUX__Pwm1A0Trig0ToQdc1Phasea

enumerator kINPUTMUX__Pwm1A0Trig1ToQdc1Phasea

enumerator kINPUTMUX_Pwm1A1Trig0ToQdc1Phasea
enumerator kINPUTMUX_Pwm1A1Trig1ToQdc1Phasea
enumerator kINPUTMUX_Pwm1A2Trig0ToQdc1Phasea
enumerator kINPUTMUX_Pwm1A2Trig1ToQdc1Phasea
enumerator kINPUTMUX_Pwm1A3Trig0ToQdc1Phasea
enumerator kINPUTMUX_Pwm1A3Trig1ToQdc1Phasea
enumerator kINPUTMUX_Qdc0CmpPosMatchToQdc1Phasea
enumerator kINPUTMUX_Qdc1CmpPosMatchToQdc1Phasea
enumerator kINPUTMUX_EvtgOut0AToQdc1Phasea
enumerator kINPUTMUX_EvtgOut0BToQdc1Phasea
enumerator kINPUTMUX_EvtgOut1AToQdc1Phasea
enumerator kINPUTMUX_EvtgOut1BToQdc1Phasea
enumerator kINPUTMUX_EvtgOut2AToQdc1Phasea
enumerator kINPUTMUX_EvtgOut2BToQdc1Phasea
enumerator kINPUTMUX_EvtgOut3AToQdc1Phasea
enumerator kINPUTMUX_EvtgOut3BToQdc1Phasea
enumerator kINPUTMUX_TrigIn0ToQdc1Phasea
enumerator kINPUTMUX_TrigIn1ToQdc1Phasea
enumerator kINPUTMUX_TrigIn2ToQdc1Phasea
enumerator kINPUTMUX_TrigIn3ToQdc1Phasea
enumerator kINPUTMUX_TrigIn4ToQdc1Phasea
enumerator kINPUTMUX_TrigIn5ToQdc1Phasea
enumerator kINPUTMUX_TrigIn6ToQdc1Phasea
enumerator kINPUTMUX_TrigIn7ToQdc1Phasea
enumerator kINPUTMUX_TrigIn8ToQdc1Phasea
enumerator kINPUTMUX_TrigIn9ToQdc1Phasea
FlexPWM0_SM0_EXTSYNC input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__ArmTxevToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A0Trig1ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A1Trig0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A1Trig1ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A2Trig0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A2Trig1ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A3Trig0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Pwm1A3Trig1ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut0AToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut0BToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut1AToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut1BToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Sm0ExtSync

enumerator kINPUTMUX__TrigIn2ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm0Sm0ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm0Sm0ExtSync
FlexPWM0_SM1_EXTSYNC input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__PinInt5ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Ctimer2M0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__ArmTxevToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Sm1ExtSync

enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm1ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm1ExtSync
FlexPWM0_SM2_EXTSYNC2 input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm2ExtSync

enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm0Sm2ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm0Sm2ExtSync
FlexPWM0_SM3_EXTSYNC input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__PinInt5ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Ctimer2M0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__ArmTxevToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A0Trig1ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A1Trig0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A1Trig1ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A2Trig0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A2Trig1ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A3Trig0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Pwm1A3Trig1ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut0AToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut0BToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut1AToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut1BToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Sm3ExtSync

enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm3ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm3ExtSync
FlexPWM0_SM0_EXTa input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_ArmTxevToFlexPwm0Sm0Exta
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Cmp0OutToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Cmp1OutToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm0Exta
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm0Exta

enumerator kINPUTMUX__EvtgOut0AToFlexPwm0Sm0Exta
enumerator kINPUTMUX__EvtgOut0BToFlexPwm0Sm0Exta
enumerator kINPUTMUX__EvtgOut1AToFlexPwm0Sm0Exta
enumerator kINPUTMUX__EvtgOut1BToFlexPwm0Sm0Exta
enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Sm0Exta
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Sm0Exta
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Sm0Exta
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm0Sm0Exta
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm0Sm0Exta
FlexPWM0_SM1_EXTa input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__PinInt5ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Ctimer2M0ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__ArmTxevToFlexPwm0Sm1Exta
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Sm1Exta

enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A0Trig1ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A1Trig0ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A1Trig1ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A2Trig0ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A2Trig1ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A3Trig0ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Pwm1A3Trig1ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm0Sm1Ext
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut0AToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut0BToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut1AToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut1BToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Sm1Ext
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Sm1Ext
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Sm1Ext
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Sm1Ext

enumerator kINPUTMUX__TrigIn6ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm0Sm1Exta
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm0Sm1Exta
FlexPWM0_SM2_EXTa input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__PinInt5ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Ctimer2M0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__ArmTxevToFlexPwm0Sm2Exta
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Pwm1A0Trig1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Pwm1A1Trig0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Pwm1A1Trig1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX__Pwm1A2Trig0ToFlexPwm0Sm2Exta

enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Sm2Exta
enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn2ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn3ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn4ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn5ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn6ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn7ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn8ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_TrigIn9ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Sm2Exta
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm2Exta
FlexPWM0_SM3_EXTa input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX_PinInt5ToFlexPwm0Sm3Exta
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Sm3Exta
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Sm3Exta
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Sm3Exta

enumerator kINPUTMUX__Ctimer2M0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__ArmTxevToFlexPwm0Sm3Exta
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A0Trig1ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A1Trig0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A1Trig1ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A2Trig0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A2Trig1ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A3Trig0ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Pwm1A3Trig1ToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm0Sm3Exta
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut0AToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut0BToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut1AToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut1BToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Sm3Exta
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Sm3Exta
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Sm3Exta

enumerator kINPUTMUX__TrigIn1ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm0Sm3ExtA
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm0Sm3ExtA
FlexPWM0_EXTFORCE input trigger connections.

enumerator kINPUTMUX__PinInt0ToFlexPwm0ExtForce
enumerator kINPUTMUX__PinInt5ToFlexPwm0ExtForce
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm0ExtForce
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm0ExtForce
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm0ExtForce
enumerator kINPUTMUX__Ctimer2M0ToFlexPwm0ExtForce
enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0ExtForce
enumerator kINPUTMUX__ArmTxevToFlexPwm0ExtForce
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0ExtForce
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0ExtForce
enumerator kINPUTMUX__Cmp0OutToFlexPwm0ExtForce
enumerator kINPUTMUX__Cmp1OutToFlexPwm0ExtForce

enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0ExtForce
enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0ExtForce
enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0ExtForce
enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0ExtForce
enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0ExtForce
enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0ExtForce
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0ExtForce
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0ExtForce
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0ExtForce
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut0AToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut0BToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut1AToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut1BToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut2AToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut2BToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut3AToFlexPwm0ExtForce
enumerator kINPUTMUX_EvtgOut3BToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn0ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn1ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn2ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn3ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn4ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn5ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn6ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn7ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn8ToFlexPwm0ExtForce
enumerator kINPUTMUX_TrigIn9ToFlexPwm0ExtForce
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0ExtForce
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0ExtForce
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0ExtForce
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0ExtForce
FlexPWM0_FAULT0 input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm0Fault0
enumerator kINPUTMUX_PinInt5ToFlexPwm0Fault0
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Fault0
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Fault0
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Fault0
enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Fault0
enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Fault0
enumerator kINPUTMUX_ArmTxevToFlexPwm0Fault0
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Fault0
enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Fault0
enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Fault0
enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Fault0
enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Fault0
enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Fault0
enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Fault0
enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Fault0
enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Fault0
enumerator kINPUTMUX_Cmp0OutToFlexPwm0Fault0
enumerator kINPUTMUX_Cmp1OutToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Fault0
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Fault0
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Fault0
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Fault0
enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Fault0
enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Fault0
enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Fault0
enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Fault0

enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Fault0
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Fault0
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Fault0
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Fault0
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Fault0
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Fault0
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm0Fault0
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm0Fault0
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm0Fault0
FlexPWM0_FAULT1 input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm0Fault1
enumerator kINPUTMUX__PinInt5ToFlexPwm0Fault1
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm0Fault1
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm0Fault1
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm0Fault1
enumerator kINPUTMUX__Ctimer2M0ToFlexPwm0Fault1
enumerator kINPUTMUX__Ctimer4M0ToFlexPwm0Fault1
enumerator kINPUTMUX__ArmTxevToFlexPwm0Fault1
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm0Fault1
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Fault1
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Fault1
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Fault1
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Fault1
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Fault1

enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Fault1
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Fault1
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Fault1
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Fault1
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A0Trig1ToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A1Trig0ToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A1Trig1ToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A2Trig0ToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A2Trig1ToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A3Trig0ToFlexPwm0Fault1
enumerator kINPUTMUX__Pwm1A3Trig1ToFlexPwm0Fault1
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm0Fault1
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut0AToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut0BToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut1AToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut1BToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Fault1
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn5ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Fault1
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Fault1

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Fault1
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Fault1
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Fault1
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault1
FlexPWM0_FAULT2 input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm0Fault2
enumerator kINPUTMUX_PinInt5ToFlexPwm0Fault2
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Fault2
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Fault2
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Fault2
enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Fault2
enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Fault2
enumerator kINPUTMUX_ArmTxevToFlexPwm0Fault2
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Fault2
enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm0Fault2
enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm0Fault2
enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm0Fault2
enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm0Fault2
enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm0Fault2
enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm0Fault2
enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm0Fault2
enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm0Fault2
enumerator kINPUTMUX_Cmp0OutToFlexPwm0Fault2
enumerator kINPUTMUX_Cmp1OutToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A0Trig0ToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A0Trig1ToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A1Trig0ToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A1Trig1ToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A2Trig0ToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A2Trig1ToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexPwm0Fault2
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexPwm0Fault2
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm0Fault2

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut0AToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut0BToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut1AToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut1BToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut2AToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut2BToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut3AToFlexPwm0Fault2
enumerator kINPUTMUX_EvtgOut3BToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn0ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn1ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn2ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn3ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn4ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn5ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn6ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn7ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn8ToFlexPwm0Fault2
enumerator kINPUTMUX_TrigIn9ToFlexPwm0Fault2
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm0Fault2
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm0Fault2
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm0Fault2
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault2
FlexPWM0_FAULT3 input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm0Fault3
enumerator kINPUTMUX_PinInt5ToFlexPwm0Fault3
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm0Fault3
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm0Fault3
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm0Fault3
enumerator kINPUTMUX_Ctimer2M0ToFlexPwm0Fault3
enumerator kINPUTMUX_Ctimer4M0ToFlexPwm0Fault3
enumerator kINPUTMUX_ArmTxevToFlexPwm0Fault3
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm0Fault3

enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm0Fault3
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm0Fault3
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm0Fault3
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm0Fault3
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm0Fault3
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm0Fault3
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm0Fault3
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm0Fault3
enumerator kINPUTMUX__Cmp0OutToFlexPwm0Fault3
enumerator kINPUTMUX__Cmp1OutToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A0Trig0ToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A0Trig1ToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A1Trig0ToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A1Trig1ToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A2Trig0ToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A2Trig1ToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A3Trig0ToFlexPwm0Fault3
enumerator kINPUTMUX__Pwm1A3Trig1ToFlexPwm0Fault3
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm0Fault3
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut0AToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut0BToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut1AToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut1BToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut2AToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut2BToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut3AToFlexPwm0Fault3
enumerator kINPUTMUX__EvtgOut3BToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn0ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn1ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn2ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn3ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn4ToFlexPwm0Fault3

enumerator kINPUTMUX__TrigIn5ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn6ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn7ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn8ToFlexPwm0Fault3
enumerator kINPUTMUX__TrigIn9ToFlexPwm0Fault3
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm0Fault3
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm0Fault3
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm0Fault3
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm0Fault3
FlexPWM1_SM0_EXTSYNC input trigger connections.

enumerator kINPUTMUX__PinInt0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__PinInt2ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__ArmTxevToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Cmp1OutToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1Sm0ExtSync

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm0ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm0ExtSync
FlexPWM1_SM1_EXTSYNC input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__ArmTxevToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Cmp1OutToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A2Trig0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A2Trig1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A3Trig0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Pwm0A3Trig1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut0AToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut0BToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut1AToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut1BToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut2AToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut2BToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut3AToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__EvtgOut3BToFlexPwm1Sm1ExtSync

enumerator kINPUTMUX__TrigIn0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn2ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn3ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn4ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn5ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn6ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn7ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn8ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__TrigIn9ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm1Sm1ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm1Sm1ExtSync
FlexPWM1_SM2_EXTSYNC2 input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__PinInt2ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__ArmTxevToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Sm2ExtSync

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm2ExtSync
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm2ExtSync
FlexPWM1_SM3_EXTSYNC input trigger connections.

enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm3ExtSync

enumerator kINPUTMUX__EvtgOut2AToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__EvtgOut2BToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__EvtgOut3AToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__EvtgOut3BToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn2ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn3ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn4ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn5ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn6ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn7ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn8ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__TrigIn9ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm1Sm3ExtSync
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm1Sm3ExtSync
FlexPWM1_SM0_EXTa input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__PinInt2ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__ArmTxevToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Sm0ExtA
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Sm0ExtA

enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Cmp1OutToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A2Trig0ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A2Trig1ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A3Trig0ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Pwm0A3Trig1ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm1Sm0Exta
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut0AToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut0BToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut1AToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut1BToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut2AToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut2BToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut3AToFlexPwm1Sm0Exta
enumerator kINPUTMUX__EvtgOut3BToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn0ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn1ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn2ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn3ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn4ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn5ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn6ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn7ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn8ToFlexPwm1Sm0Exta
enumerator kINPUTMUX__TrigIn9ToFlexPwm1Sm0Exta

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm0Exta
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm0Exta
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm0Exta
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm0Exta
FlexPWM1_SM1_EXTa input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm1Exta
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Cmp0OutToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Cmp1OutToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm1Exta
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm1Exta

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm1ExtA
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm1ExtA
FlexPWM1_SM2_EXT_A input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_PinInt2ToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_ArmTxevToFlexPwm1Sm2ExtA
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Sm2ExtA

enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Cmp1OutToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A2Trig0ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A2Trig1ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A3Trig0ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Pwm0A3Trig1ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm1Sm2Ext
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut0AToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut0BToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut1AToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut1BToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut2AToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut2BToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut3AToFlexPwm1Sm2Ext
enumerator kINPUTMUX__EvtgOut3BToFlexPwm1Sm2Ext
enumerator kINPUTMUX__TrigIn0ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__TrigIn1ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__TrigIn2ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__TrigIn3ToFlexPwm1Sm2Ext
enumerator kINPUTMUX__TrigIn4ToFlexPwm1Sm2Ext

enumerator kINPUTMUX__TrigIn5ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__TrigIn6ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__TrigIn7ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__TrigIn8ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__TrigIn9ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm1Sm2Exta
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm1Sm2Exta
FlexPWM1_SM3_EXTa input trigger connections.

enumerator kINPUTMUX__PinInt0ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__PinInt2ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__ArmTxevToFlexPwm1Sm3Exta
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Cmp1OutToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1Sm3Exta
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1Sm3Exta

enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn0ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn1ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn2ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn3ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn4ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn5ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn6ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn7ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn8ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_TrigIn9ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Sm3ExtA
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm3ExtA
FlexPWM1_EXTFORCE input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm1ExtForce
enumerator kINPUTMUX_PinInt2ToFlexPwm1ExtForce
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1ExtForce
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1ExtForce

enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1ExtForce
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1ExtForce
enumerator kINPUTMUX__ArmTxevToFlexPwm1ExtForce
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1ExtForce
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1ExtForce
enumerator kINPUTMUX__Cmp0OutToFlexPwm1ExtForce
enumerator kINPUTMUX__Cmp1OutToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A2Trig0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A2Trig1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A3Trig0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Pwm0A3Trig1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm1ExtForce
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut0AToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut0BToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut1AToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut1BToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut2AToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut2BToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut3AToFlexPwm1ExtForce
enumerator kINPUTMUX__EvtgOut3BToFlexPwm1ExtForce

enumerator kINPUTMUX__TrigIn0ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn1ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn2ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn3ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn4ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn5ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn6ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn7ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn8ToFlexPwm1ExtForce
enumerator kINPUTMUX__TrigIn9ToFlexPwm1ExtForce
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm1ExtForce
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm1ExtForce
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm1ExtForce
FlexPWM1_FAULT0 input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm1Fault0
enumerator kINPUTMUX__PinInt2ToFlexPwm1Fault0
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm1Fault0
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm1Fault0
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Fault0
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Fault0
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Fault0
enumerator kINPUTMUX__ArmTxevToFlexPwm1Fault0
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Fault0
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Fault0
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Fault0
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Fault0
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Fault0
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Fault0
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Fault0
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Fault0
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Fault0
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Fault0

enumerator kINPUTMUX_Cmp1OutToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Fault0
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Fault0
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Fault0
enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Fault0
enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn0ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn1ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn2ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn3ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn4ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn5ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn6ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn7ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn8ToFlexPwm1Fault0
enumerator kINPUTMUX_TrigIn9ToFlexPwm1Fault0
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Fault0
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Fault0
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Fault0
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault0
FlexPWM1_FAULT1 input trigger connections.

enumerator kINPUTMUX__PinInt0ToFlexPwm1Fault1
enumerator kINPUTMUX__PinInt2ToFlexPwm1Fault1
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm1Fault1
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm1Fault1
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Fault1
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Fault1
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Fault1
enumerator kINPUTMUX__ArmTxevToFlexPwm1Fault1
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Fault1
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Fault1
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Fault1
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Fault1
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Fault1
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Fault1
enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Fault1
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Fault1
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Fault1
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Fault1
enumerator kINPUTMUX__Cmp1OutToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A2Trig0ToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A2Trig1ToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A3Trig0ToFlexPwm1Fault1
enumerator kINPUTMUX__Pwm0A3Trig1ToFlexPwm1Fault1
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm1Fault1
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm1Fault1
enumerator kINPUTMUX__EvtgOut0AToFlexPwm1Fault1
enumerator kINPUTMUX__EvtgOut0BToFlexPwm1Fault1
enumerator kINPUTMUX__EvtgOut1AToFlexPwm1Fault1
enumerator kINPUTMUX__EvtgOut1BToFlexPwm1Fault1

enumerator kINPUTMUX__EvtgOut2AToFlexPwm1Fault1
enumerator kINPUTMUX__EvtgOut2BToFlexPwm1Fault1
enumerator kINPUTMUX__EvtgOut3AToFlexPwm1Fault1
enumerator kINPUTMUX__EvtgOut3BToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn0ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn1ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn2ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn3ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn4ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn5ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn6ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn7ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn8ToFlexPwm1Fault1
enumerator kINPUTMUX__TrigIn9ToFlexPwm1Fault1
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexPwm1Fault1
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexPwm1Fault1
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexPwm1Fault1
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexPwm1Fault1
FlexPWM1_FAULT2 input trigger connections.
enumerator kINPUTMUX__PinInt0ToFlexPwm1Fault2
enumerator kINPUTMUX__PinInt2ToFlexPwm1Fault2
enumerator kINPUTMUX__Ctimer0M3ToFlexPwm1Fault2
enumerator kINPUTMUX__Ctimer1M3ToFlexPwm1Fault2
enumerator kINPUTMUX__Ctimer2M3ToFlexPwm1Fault2
enumerator kINPUTMUX__Ctimer2M1ToFlexPwm1Fault2
enumerator kINPUTMUX__Ctimer4M1ToFlexPwm1Fault2
enumerator kINPUTMUX__ArmTxevToFlexPwm1Fault2
enumerator kINPUTMUX__GpioIntBmatchToFlexPwm1Fault2
enumerator kINPUTMUX__Adc0Tcomp0ToFlexPwm1Fault2
enumerator kINPUTMUX__Adc0Tcomp1ToFlexPwm1Fault2
enumerator kINPUTMUX__Adc0Tcomp2ToFlexPwm1Fault2
enumerator kINPUTMUX__Adc0Tcomp3ToFlexPwm1Fault2
enumerator kINPUTMUX__Adc1Tcomp0ToFlexPwm1Fault2

enumerator kINPUTMUX__Adc1Tcomp1ToFlexPwm1Fault2
enumerator kINPUTMUX__Adc1Tcomp2ToFlexPwm1Fault2
enumerator kINPUTMUX__Adc1Tcomp3ToFlexPwm1Fault2
enumerator kINPUTMUX__Cmp0OutToFlexPwm1Fault2
enumerator kINPUTMUX__Cmp1OutToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A0Trig0ToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A0Trig1ToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A1Trig0ToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A1Trig1ToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A2Trig0ToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A2Trig1ToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A3Trig0ToFlexPwm1Fault2
enumerator kINPUTMUX__Pwm0A3Trig1ToFlexPwm1Fault2
enumerator kINPUTMUX__Qdc0CmpPosMatchToFlexPwm1Fault2
enumerator kINPUTMUX__Qdc1CmpPosMatchToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut0AToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut0BToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut1AToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut1BToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut2AToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut2BToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut3AToFlexPwm1Fault2
enumerator kINPUTMUX__EvtgOut3BToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn0ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn1ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn2ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn3ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn4ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn5ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn6ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn7ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn8ToFlexPwm1Fault2
enumerator kINPUTMUX__TrigIn9ToFlexPwm1Fault2

enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Fault2
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Fault2
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Fault2
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault2
FlexPWM1_FAULT3 input trigger connections.
enumerator kINPUTMUX_PinInt0ToFlexPwm1Fault3
enumerator kINPUTMUX_PinInt2ToFlexPwm1Fault3
enumerator kINPUTMUX_Ctimer0M3ToFlexPwm1Fault3
enumerator kINPUTMUX_Ctimer1M3ToFlexPwm1Fault3
enumerator kINPUTMUX_Ctimer2M3ToFlexPwm1Fault3
enumerator kINPUTMUX_Ctimer2M1ToFlexPwm1Fault3
enumerator kINPUTMUX_Ctimer4M1ToFlexPwm1Fault3
enumerator kINPUTMUX_ArmTxevToFlexPwm1Fault3
enumerator kINPUTMUX_GpioIntBmatchToFlexPwm1Fault3
enumerator kINPUTMUX_Adc0Tcomp0ToFlexPwm1Fault3
enumerator kINPUTMUX_Adc0Tcomp1ToFlexPwm1Fault3
enumerator kINPUTMUX_Adc0Tcomp2ToFlexPwm1Fault3
enumerator kINPUTMUX_Adc0Tcomp3ToFlexPwm1Fault3
enumerator kINPUTMUX_Adc1Tcomp0ToFlexPwm1Fault3
enumerator kINPUTMUX_Adc1Tcomp1ToFlexPwm1Fault3
enumerator kINPUTMUX_Adc1Tcomp2ToFlexPwm1Fault3
enumerator kINPUTMUX_Adc1Tcomp3ToFlexPwm1Fault3
enumerator kINPUTMUX_Cmp0OutToFlexPwm1Fault3
enumerator kINPUTMUX_Cmp1OutToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A0Trig0ToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A0Trig1ToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A1Trig0ToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A1Trig1ToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A2Trig0ToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexPwm1Fault3
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexPwm1Fault3
enumerator kINPUTMUX_Qdc0CmpPosMatchToFlexPwm1Fault3

enumerator kINPUTMUX_Qdc1CmpPosMatchToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut0AToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut0BToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut1AToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut1BToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut2AToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut2BToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut3AToFlexPwm1Fault3
enumerator kINPUTMUX_EvtgOut3BToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn0ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn1ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn2ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn3ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn4ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn5ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn6ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn7ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn8ToFlexPwm1Fault3
enumerator kINPUTMUX_TrigIn9ToFlexPwm1Fault3
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexPwm1Fault3
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexPwm1Fault3
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexPwm1Fault3
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault3
PWM0 external clock trigger.
enumerator kINPUTMUX_Fro16KToPwm0ExtClk
enumerator kINPUTMUX_Osc32KToPwm0ExtClk
enumerator kINPUTMUX_EvtgOut0AToPwm0ExtClk
enumerator kINPUTMUX_EvtgOut1AToPwm0ExtClk
enumerator kINPUTMUX_ExttrigIn0ToPwm0ExtClk
enumerator kINPUTMUX_ExttrigIn1ToPwm0ExtClk
PWM1 external clock trigger.
enumerator kINPUTMUX_Fro16KToPwm1ExtClk
enumerator kINPUTMUX_Osc32KToPwm1ExtClk
enumerator kINPUTMUX_EvtgOut0AToPwm1ExtClk

enumerator kINPUTMUX__EvtgOut1AToPwm1ExtClk

enumerator kINPUTMUX__ExttrigIn0ToPwm1ExtClk

enumerator kINPUTMUX__ExttrigIn1ToPwm1ExtClk

EVTG trigger input connections.

enumerator kINPUTMUX__PinInt0ToEvtgTrigger

enumerator kINPUTMUX__PinInt1ToEvtgTrigger

enumerator kINPUTMUX__Ctimer0M3ToEvtgTrigger

enumerator kINPUTMUX__Ctimer1M3ToEvtgTrigger

enumerator kINPUTMUX__Ctimer2M3ToEvtgTrigger

enumerator kINPUTMUX__Ctimer2M2ToEvtgTrigger

enumerator kINPUTMUX__Ctimer3M2ToEvtgTrigger

enumerator kINPUTMUX__Ctimer4M2ToEvtgTrigger

enumerator kINPUTMUX__GpioIntBmatchToEvtgTrigger

enumerator kINPUTMUX__Adc0IrqToEvtgTrigger

enumerator kINPUTMUX__Adc1IrqToEvtgTrigger

enumerator kINPUTMUX__Adc0Tcomp0ToEvtgTrigger

enumerator kINPUTMUX__Adc0Tcomp1ToEvtgTrigger

enumerator kINPUTMUX__Adc0Tcomp2ToEvtgTrigger

enumerator kINPUTMUX__Adc0Tcomp3ToEvtgTrigger

enumerator kINPUTMUX__Adc1Tcomp0ToEvtgTrigger

enumerator kINPUTMUX__Adc1Tcomp1ToEvtgTrigger

enumerator kINPUTMUX__Adc1Tcomp2ToEvtgTrigger

enumerator kINPUTMUX__Adc1Tcomp3ToEvtgTrigger

enumerator kINPUTMUX__Cmp0OutToEvtgTrigger

enumerator kINPUTMUX__Cmp1OutToEvtgTrigger

enumerator kINPUTMUX__Pwm0A0Trig0ToEvtgTrigger

enumerator kINPUTMUX__Pwm0A0Trig1ToEvtgTrigger

enumerator kINPUTMUX__Pwm0A1Trig0ToEvtgTrigger

enumerator kINPUTMUX__Pwm0A1Trig1ToEvtgTrigger

enumerator kINPUTMUX__Pwm0A2Trig0ToEvtgTrigger

enumerator kINPUTMUX__Pwm0A2Trig1ToEvtgTrigger

enumerator kINPUTMUX__Pwm0A3Trig0ToEvtgTrigger

enumerator kINPUTMUX__Pwm0A3Trig1ToEvtgTrigger

enumerator kINPUTMUX_Pwm1A0Trig0ToEvtgTrigger
enumerator kINPUTMUX_Pwm1A0Trig1ToEvtgTrigger
enumerator kINPUTMUX_Pwm1A1Trig0ToEvtgTrigger
enumerator kINPUTMUX_Pwm1A1Trig1ToEvtgTrigger
enumerator kINPUTMUX_Pwm1A2Trig0ToEvtgTrigger
enumerator kINPUTMUX_Pwm1A2Trig1ToEvtgTrigger
enumerator kINPUTMUX_Pwm1A3Trig0ToEvtgTrigger
enumerator kINPUTMUX_Pwm1A3Trig1ToEvtgTrigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToEvtgTrigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToEvtgTrigger
enumerator kINPUTMUX_TrigIn0ToEvtgTrigger
enumerator kINPUTMUX_TrigIn1ToEvtgTrigger
enumerator kINPUTMUX_TrigIn2ToEvtgTrigger
enumerator kINPUTMUX_TrigIn3ToEvtgTrigger
enumerator kINPUTMUX_Lptmr0ToEvtgTrigger
enumerator kINPUTMUX_Lptmr1ToEvtgTrigger

EXT trigger connections.

enumerator kINPUTMUX_PinInt0ToExtTrigger
enumerator kINPUTMUX_PinInt1ToExtTrigger
enumerator kINPUTMUX_Adc0IrqToExtTrigger
enumerator kINPUTMUX_Adc1IrqToExtTrigger
enumerator kINPUTMUX_Adc0Tcomp0ToExtTrigger
enumerator kINPUTMUX_Adc1Tcomp0ToExtTrigger
enumerator kINPUTMUX_Pwm0A0Trig01ToExtTrigger
enumerator kINPUTMUX_Pwm0A1Trig01ToExtTrigger
enumerator kINPUTMUX_Pwm0A2Trig01ToExtTrigger
enumerator kINPUTMUX_Pwm0A3Trig01ToExtTrigger
enumerator kINPUTMUX_Pwm1A0Trig01ToExtTrigger
enumerator kINPUTMUX_Pwm1A1Trig01ToExtTrigger
enumerator kINPUTMUX_Pwm1A2Trig01ToExtTrigger
enumerator kINPUTMUX_Pwm1A3Trig01ToExtTrigger
enumerator kINPUTMUX_Qdc0CmpPosMatchToExtTrigger
enumerator kINPUTMUX_Qdc1CmpPosMatchToExtTrigger

enumerator kINPUTMUX__EvtgOut0AToExtTrigger
enumerator kINPUTMUX__EvtgOut0BToExtTrigger
enumerator kINPUTMUX__EvtgOut1AToExtTrigger
enumerator kINPUTMUX__EvtgOut1BToExtTrigger
enumerator kINPUTMUX__EvtgOut2AToExtTrigger
enumerator kINPUTMUX__EvtgOut2BToExtTrigger
enumerator kINPUTMUX__EvtgOut3AToExtTrigger
enumerator kINPUTMUX__EvtgOut3BToExtTrigger
enumerator kINPUTMUX__Lptmr0ToExtTrigger
enumerator kINPUTMUX__Lptmr1ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm0Trig3ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm1Trig3ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm2Trig3ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm3Trig3ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm4Trig3ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm5Trig3ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm6Trig3ToExtTrigger
enumerator kINPUTMUX__Lpflexcomm7Trig3ToExtTrigger
enumerator kINPUTMUX__Cmp0OutToExtTrigger
enumerator kINPUTMUX__Cmp1OutToExtTrigger

FLEXCOMM0 trigger input connections.

enumerator kINPUTMUX__PinInt4ToFlexcomm0Trigger
enumerator kINPUTMUX__PinInt5ToFlexcomm0Trigger
enumerator kINPUTMUX__PinInt6ToFlexcomm0Trigger
enumerator kINPUTMUX__Ctimer0M1ToFlexcomm0Trigger
enumerator kINPUTMUX__Ctimer1M1ToFlexcomm0Trigger
enumerator kINPUTMUX__Ctimer2M0ToFlexcomm0Trigger
enumerator kINPUTMUX__Ctimer3M0ToFlexcomm0Trigger
enumerator kINPUTMUX__Ctimer4M0ToFlexcomm0Trigger
enumerator kINPUTMUX__Lptmr0ToFlexcomm0Trigger
enumerator kINPUTMUX__Lptmr1ToFlexcomm0Trigger
enumerator kINPUTMUX__ArmTxevToFlexcomm0Trigger
enumerator kINPUTMUX__GpioIntBmatchToFlexcomm0Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm0Trigger
enumerator kINPUTMUX_Cmp1OutToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut0AToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut0BToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut1AToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut1BToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut2AToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut2BToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut3AToFlexcomm0Trigger
enumerator kINPUTMUX_EvtgOut3BToFlexcomm0Trigger
enumerator kINPUTMUX_TrigIn0ToFlexcomm0Trigger
enumerator kINPUTMUX_TrigIn1ToFlexcomm0Trigger
enumerator kINPUTMUX_TrigIn2ToFlexcomm0Trigger
enumerator kINPUTMUX_TrigIn3ToFlexcomm0Trigger
enumerator kINPUTMUX_TrigIn4ToFlexcomm0Trigger
enumerator kINPUTMUX_TrigIn10ToFlexcomm0Trigger
enumerator kINPUTMUX_TrigIn11ToFlexcomm0Trigger
enumerator kINPUTMUX_FlexioCh4ToFlexcomm0Trigger
enumerator kINPUTMUX_FlexioCh5ToFlexcomm0Trigger
enumerator kINPUTMUX_FlexioCh6ToFlexcomm0Trigger
enumerator kINPUTMUX_FlexioCh7ToFlexcomm0Trigger
enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm0Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm0Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm0Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm0Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm0Trigger
enumerator kINPUTMUX_WuuToFlexcomm0Trigger
 FLEXCOMM1 trigger input connections.
enumerator kINPUTMUX_PinInt4ToFlexcomm1Trigger
enumerator kINPUTMUX_PinInt5ToFlexcomm1Trigger
enumerator kINPUTMUX_PinInt6ToFlexcomm1Trigger
enumerator kINPUTMUX_Ctimer0M1ToFlexcomm1Trigger
enumerator kINPUTMUX_Ctimer1M1ToFlexcomm1Trigger

enumerator kINPUTMUX__Ctimer2M0ToFlexcomm1Trigger
enumerator kINPUTMUX__Ctimer3M0ToFlexcomm1Trigger
enumerator kINPUTMUX__Ctimer4M0ToFlexcomm1Trigger
enumerator kINPUTMUX__Lptmr0ToFlexcomm1Trigger
enumerator kINPUTMUX__Lptmr1ToFlexcomm1Trigger
enumerator kINPUTMUX__ArmTxevToFlexcomm1Trigger
enumerator kINPUTMUX__GpioIntBmatchToFlexcomm1Trigger
enumerator kINPUTMUX__Cmp0OutToFlexcomm1Trigger
enumerator kINPUTMUX__Cmp1OutToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut0AToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut0BToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut1AToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut1BToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut2AToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut2BToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut3AToFlexcomm1Trigger
enumerator kINPUTMUX__EvtgOut3BToFlexcomm1Trigger
enumerator kINPUTMUX__TrigIn0ToFlexcomm1Trigger
enumerator kINPUTMUX__TrigIn1ToFlexcomm1Trigger
enumerator kINPUTMUX__TrigIn2ToFlexcomm1Trigger
enumerator kINPUTMUX__TrigIn3ToFlexcomm1Trigger
enumerator kINPUTMUX__TrigIn4ToFlexcomm1Trigger
enumerator kINPUTMUX__TrigIn10ToFlexcomm1Trigger
enumerator kINPUTMUX__TrigIn11ToFlexcomm1Trigger
enumerator kINPUTMUX__FlexioCh4ToFlexcomm1Trigger
enumerator kINPUTMUX__FlexioCh5ToFlexcomm1Trigger
enumerator kINPUTMUX__FlexioCh6ToFlexcomm1Trigger
enumerator kINPUTMUX__FlexioCh7ToFlexcomm1Trigger
enumerator kINPUTMUX__Usb0IppIndUartRxdUsbmuxToFlexcomm1Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexcomm1Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexcomm1Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexcomm1Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexcomm1Trigger

enumerator kINPUTMUX__WuuToFlexcomm1Trigger
FLEXCOMM2 trigger input connections.

enumerator kINPUTMUX__PinInt4ToFlexcomm2Trigger

enumerator kINPUTMUX__PinInt6ToFlexcomm2Trigger

enumerator kINPUTMUX__PinInt7ToFlexcomm2Trigger

enumerator kINPUTMUX__Ctimer0M1ToFlexcomm2Trigger

enumerator kINPUTMUX__Ctimer1M1ToFlexcomm2Trigger

enumerator kINPUTMUX__Ctimer2M1ToFlexcomm2Trigger

enumerator kINPUTMUX__Ctimer3M1ToFlexcomm2Trigger

enumerator kINPUTMUX__Ctimer4M1ToFlexcomm2Trigger

enumerator kINPUTMUX__Lptmr0ToFlexcomm2Trigger

enumerator kINPUTMUX__Lptmr1ToFlexcomm2Trigger

enumerator kINPUTMUX__ArmTxevToFlexcomm2Trigger

enumerator kINPUTMUX__GpioIntBmatchToFlexcomm2Trigger

enumerator kINPUTMUX__Cmp0OutToFlexcomm2Trigger

enumerator kINPUTMUX__Cmp1OutToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut0AToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut0BToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut1AToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut1BToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut2AToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut2BToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut3AToFlexcomm2Trigger

enumerator kINPUTMUX__EvtgOut3BToFlexcomm2Trigger

enumerator kINPUTMUX__TrigIn0ToFlexcomm2Trigger

enumerator kINPUTMUX__TrigIn1ToFlexcomm2Trigger

enumerator kINPUTMUX__TrigIn2ToFlexcomm2Trigger

enumerator kINPUTMUX__TrigIn3ToFlexcomm2Trigger

enumerator kINPUTMUX__TrigIn4ToFlexcomm2Trigger

enumerator kINPUTMUX__TrigIn10ToFlexcomm2Trigger

enumerator kINPUTMUX__TrigIn11ToFlexcomm2Trigger

enumerator kINPUTMUX__FlexioCh4ToFlexcomm2Trigger

enumerator kINPUTMUX__FlexioCh5ToFlexcomm2Trigger

enumerator kINPUTMUX_FlexioCh6ToFlexcomm2Trigger
enumerator kINPUTMUX_FlexioCh7ToFlexcomm2Trigger
enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm2Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm2Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm2Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm2Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm2Trigger
enumerator kINPUTMUX_WuuToFlexcomm2Trigger

FLEXCOMM3 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm3Trigger
enumerator kINPUTMUX_PinInt5ToFlexcomm3Trigger
enumerator kINPUTMUX_PinInt7ToFlexcomm3Trigger
enumerator kINPUTMUX_Ctimer0M1ToFlexcomm3Trigger
enumerator kINPUTMUX_Ctimer1M1ToFlexcomm3Trigger
enumerator kINPUTMUX_Ctimer2M1ToFlexcomm3Trigger
enumerator kINPUTMUX_Ctimer3M1ToFlexcomm3Trigger
enumerator kINPUTMUX_Ctimer4M1ToFlexcomm3Trigger
enumerator kINPUTMUX_Lptmr0ToFlexcomm3Trigger
enumerator kINPUTMUX_Lptmr1ToFlexcomm3Trigger
enumerator kINPUTMUX_ArmTxevToFlexcomm3Trigger
enumerator kINPUTMUX_GpioIntBmatchToFlexcomm3Trigger
enumerator kINPUTMUX_Cmp0OutToFlexcomm3Trigger
enumerator kINPUTMUX_Cmp1OutToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut0AToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut0BToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut1AToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut1BToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut2AToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut2BToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut3AToFlexcomm3Trigger
enumerator kINPUTMUX_EvtgOut3BToFlexcomm3Trigger
enumerator kINPUTMUX_TrigIn0ToFlexcomm3Trigger
enumerator kINPUTMUX_TrigIn1ToFlexcomm3Trigger

enumerator kINPUTMUX__TrigIn2ToFlexcomm3Trigger
enumerator kINPUTMUX__TrigIn3ToFlexcomm3Trigger
enumerator kINPUTMUX__TrigIn4ToFlexcomm3Trigger
enumerator kINPUTMUX__TrigIn10ToFlexcomm3Trigger
enumerator kINPUTMUX__TrigIn11ToFlexcomm3Trigger
enumerator kINPUTMUX__FlexioCh4ToFlexcomm3Trigger
enumerator kINPUTMUX__FlexioCh5ToFlexcomm3Trigger
enumerator kINPUTMUX__FlexioCh6ToFlexcomm3Trigger
enumerator kINPUTMUX__FlexioCh7ToFlexcomm3Trigger
enumerator kINPUTMUX__Usb0IppIndUartRxdUsbmuxToFlexcomm3Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexcomm3Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexcomm3Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexcomm3Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexcomm3Trigger
enumerator kINPUTMUX__WuuToFlexcomm3Trigger
 FLEXCOMM4 trigger input connections.
enumerator kINPUTMUX__PinInt4ToFlexcomm4Trigger
enumerator kINPUTMUX__PinInt5ToFlexcomm4Trigger
enumerator kINPUTMUX__PinInt7ToFlexcomm4Trigger
enumerator kINPUTMUX__Ctimer0M1ToFlexcomm4Trigger
enumerator kINPUTMUX__Ctimer1M1ToFlexcomm4Trigger
enumerator kINPUTMUX__Ctimer2M2ToFlexcomm4Trigger
enumerator kINPUTMUX__Ctimer3M2ToFlexcomm4Trigger
enumerator kINPUTMUX__Ctimer4M2ToFlexcomm4Trigger
enumerator kINPUTMUX__Lptmr0ToFlexcomm4Trigger
enumerator kINPUTMUX__Lptmr1ToFlexcomm4Trigger
enumerator kINPUTMUX__ArmTxevToFlexcomm4Trigger
enumerator kINPUTMUX__GpioIntBmatchToFlexcomm4Trigger
enumerator kINPUTMUX__Cmp0OutToFlexcomm4Trigger
enumerator kINPUTMUX__Cmp1OutToFlexcomm4Trigger
enumerator kINPUTMUX__EvtgOut0AToFlexcomm4Trigger
enumerator kINPUTMUX__EvtgOut0BToFlexcomm4Trigger
enumerator kINPUTMUX__EvtgOut1AToFlexcomm4Trigger

enumerator kINPUTMUX__EvtgOut1BToFlexcomm4Trigger
enumerator kINPUTMUX__EvtgOut2AToFlexcomm4Trigger
enumerator kINPUTMUX__EvtgOut2BToFlexcomm4Trigger
enumerator kINPUTMUX__EvtgOut3AToFlexcomm4Trigger
enumerator kINPUTMUX__EvtgOut3BToFlexcomm4Trigger
enumerator kINPUTMUX__TrigIn0ToFlexcomm4Trigger
enumerator kINPUTMUX__TrigIn1ToFlexcomm4Trigger
enumerator kINPUTMUX__TrigIn2ToFlexcomm4Trigger
enumerator kINPUTMUX__TrigIn3ToFlexcomm4Trigger
enumerator kINPUTMUX__TrigIn4ToFlexcomm4Trigger
enumerator kINPUTMUX__TrigIn10ToFlexcomm4Trigger
enumerator kINPUTMUX__TrigIn11ToFlexcomm4Trigger
enumerator kINPUTMUX__FlexioCh4ToFlexcomm4Trigger
enumerator kINPUTMUX__FlexioCh5ToFlexcomm4Trigger
enumerator kINPUTMUX__FlexioCh6ToFlexcomm4Trigger
enumerator kINPUTMUX__FlexioCh7ToFlexcomm4Trigger
enumerator kINPUTMUX__Usb0IppIndUartRxdUsbmuxToFlexcomm4Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexcomm4Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexcomm4Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexcomm4Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexcomm4Trigger
enumerator kINPUTMUX__WuuToFlexcomm4Trigger

FLEXCOMM5 trigger input connections.

enumerator kINPUTMUX__PinInt4ToFlexcomm5Trigger
enumerator kINPUTMUX__PinInt5ToFlexcomm5Trigger
enumerator kINPUTMUX__PinInt7ToFlexcomm5Trigger
enumerator kINPUTMUX__Ctimer0M1ToFlexcomm5Trigger
enumerator kINPUTMUX__Ctimer1M1ToFlexcomm5Trigger
enumerator kINPUTMUX__Ctimer2M2ToFlexcomm5Trigger
enumerator kINPUTMUX__Ctimer3M2ToFlexcomm5Trigger
enumerator kINPUTMUX__Ctimer4M2ToFlexcomm5Trigger
enumerator kINPUTMUX__Lptmr0ToFlexcomm5Trigger
enumerator kINPUTMUX__Lptmr1ToFlexcomm5Trigger

enumerator kINPUTMUX__ArmTxeVToFlexcomm5Trigger
enumerator kINPUTMUX__GpioIntBmatchToFlexcomm5Trigger
enumerator kINPUTMUX__Cmp0OutToFlexcomm5Trigger
enumerator kINPUTMUX__Cmp1OutToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut0AToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut0BToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut1AToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut1BToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut2AToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut2BToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut3AToFlexcomm5Trigger
enumerator kINPUTMUX__EvtgOut3BToFlexcomm5Trigger
enumerator kINPUTMUX__TrigIn0ToFlexcomm5Trigger
enumerator kINPUTMUX__TrigIn1ToFlexcomm5Trigger
enumerator kINPUTMUX__TrigIn2ToFlexcomm5Trigger
enumerator kINPUTMUX__TrigIn3ToFlexcomm5Trigger
enumerator kINPUTMUX__TrigIn4ToFlexcomm5Trigger
enumerator kINPUTMUX__TrigIn10ToFlexcomm5Trigger
enumerator kINPUTMUX__TrigIn11ToFlexcomm5Trigger
enumerator kINPUTMUX__FlexioCh4ToFlexcomm5Trigger
enumerator kINPUTMUX__FlexioCh5ToFlexcomm5Trigger
enumerator kINPUTMUX__FlexioCh6ToFlexcomm5Trigger
enumerator kINPUTMUX__FlexioCh7ToFlexcomm5Trigger
enumerator kINPUTMUX__Usb0IppIndUartRxdUsbmuxToFlexcomm5Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexcomm5Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexcomm5Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig0ToFlexcomm5Trigger
enumerator kINPUTMUX__Gpio3PinEventTrig1ToFlexcomm5Trigger
enumerator kINPUTMUX__WuuToFlexcomm5Trigger
 FLEXCOMM6 trigger input connections.
enumerator kINPUTMUX__PinInt4ToFlexcomm6Trigger
enumerator kINPUTMUX__PinInt5ToFlexcomm6Trigger
enumerator kINPUTMUX__PinInt7ToFlexcomm6Trigger

enumerator kINPUTMUX__Ctimer0M1ToFlexcomm6Trigger
enumerator kINPUTMUX__Ctimer1M1ToFlexcomm6Trigger
enumerator kINPUTMUX__Ctimer2M3ToFlexcomm6Trigger
enumerator kINPUTMUX__Ctimer3M3ToFlexcomm6Trigger
enumerator kINPUTMUX__Ctimer4M3ToFlexcomm6Trigger
enumerator kINPUTMUX__Lptmr0ToFlexcomm6Trigger
enumerator kINPUTMUX__Lptmr1ToFlexcomm6Trigger
enumerator kINPUTMUX__ArmTxevToFlexcomm6Trigger
enumerator kINPUTMUX__GpioIntBmatchToFlexcomm6Trigger
enumerator kINPUTMUX__Cmp0OutToFlexcomm6Trigger
enumerator kINPUTMUX__Cmp1OutToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut0AToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut0BToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut1AToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut1BToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut2AToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut2BToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut3AToFlexcomm6Trigger
enumerator kINPUTMUX__EvtgOut3BToFlexcomm6Trigger
enumerator kINPUTMUX__TrigIn0ToFlexcomm6Trigger
enumerator kINPUTMUX__TrigIn1ToFlexcomm6Trigger
enumerator kINPUTMUX__TrigIn2ToFlexcomm6Trigger
enumerator kINPUTMUX__TrigIn3ToFlexcomm6Trigger
enumerator kINPUTMUX__TrigIn4ToFlexcomm6Trigger
enumerator kINPUTMUX__TrigIn10ToFlexcomm6Trigger
enumerator kINPUTMUX__TrigIn11ToFlexcomm6Trigger
enumerator kINPUTMUX__FlexioCh4ToFlexcomm6Trigger
enumerator kINPUTMUX__FlexioCh5ToFlexcomm6Trigger
enumerator kINPUTMUX__FlexioCh6ToFlexcomm6Trigger
enumerator kINPUTMUX__FlexioCh7ToFlexcomm6Trigger
enumerator kINPUTMUX__Usb0IppIndUartRxdUsbmuxToFlexcomm6Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig0ToFlexcomm6Trigger
enumerator kINPUTMUX__Gpio2PinEventTrig1ToFlexcomm6Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm6Trigger

enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm6Trigger

enumerator kINPUTMUX_WuuToFlexcomm6Trigger

FLEXCOMM7 trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexcomm7Trigger

enumerator kINPUTMUX_PinInt5ToFlexcomm7Trigger

enumerator kINPUTMUX_PinInt7ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer0M1ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer1M1ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer2M3ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer3M3ToFlexcomm7Trigger

enumerator kINPUTMUX_Ctimer4M3ToFlexcomm7Trigger

enumerator kINPUTMUX_Lptmr0ToFlexcomm7Trigger

enumerator kINPUTMUX_Lptmr1ToFlexcomm7Trigger

enumerator kINPUTMUX_ArmTxevToFlexcomm7Trigger

enumerator kINPUTMUX_GpioIntBmatchToFlexcomm7Trigger

enumerator kINPUTMUX_Cmp0OutToFlexcomm7Trigger

enumerator kINPUTMUX_Cmp1OutToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut0AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut0BToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut1AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut1BToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut2AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut2BToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut3AToFlexcomm7Trigger

enumerator kINPUTMUX_EvtgOut3BToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn0ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn1ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn2ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn3ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn4ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn10ToFlexcomm7Trigger

enumerator kINPUTMUX_TrigIn11ToFlexcomm7Trigger

enumerator kINPUTMUX_FlexioCh4ToFlexcomm7Trigger
enumerator kINPUTMUX_FlexioCh5ToFlexcomm7Trigger
enumerator kINPUTMUX_FlexioCh6ToFlexcomm7Trigger
enumerator kINPUTMUX_FlexioCh7ToFlexcomm7Trigger
enumerator kINPUTMUX_Usb0IppIndUartRxdUsbmuxToFlexcomm7Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig0ToFlexcomm7Trigger
enumerator kINPUTMUX_Gpio2PinEventTrig1ToFlexcomm7Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig0ToFlexcomm7Trigger
enumerator kINPUTMUX_Gpio3PinEventTrig1ToFlexcomm7Trigger
enumerator kINPUTMUX_WuuToFlexcomm7Trigger

FlexIO trigger input connections.

enumerator kINPUTMUX_PinInt4ToFlexioTrigger
enumerator kINPUTMUX_PinInt5ToFlexioTrigger
enumerator kINPUTMUX_PinInt6ToFlexioTrigger
enumerator kINPUTMUX_PinInt7ToFlexioTrigger
enumerator kINPUTMUX_Ctimer0M1ToFlexioTrigger
enumerator kINPUTMUX_Ctimer1M1ToFlexioTrigger
enumerator kINPUTMUX_Ctimer2M1ToFlexioTrigger
enumerator kINPUTMUX_Ctimer3M1ToFlexioTrigger
enumerator kINPUTMUX_Ctimer4M1ToFlexioTrigger
enumerator kINPUTMUX_Lptmr0ToFlexioTrigger
enumerator kINPUTMUX_Lptmr1ToFlexioTrigger
enumerator kINPUTMUX_ArmTxevToFlexioTrigger
enumerator kINPUTMUX_GpioIntBmatchToFlexioTrigger
enumerator kINPUTMUX_Adc0Tcomp0ToFlexioTrigger
enumerator kINPUTMUX_Adc0Tcomp1ToFlexioTrigger
enumerator kINPUTMUX_Adc0Tcomp2ToFlexioTrigger
enumerator kINPUTMUX_Adc0Tcomp3ToFlexioTrigger
enumerator kINPUTMUX_Adc1Tcomp0ToFlexioTrigger
enumerator kINPUTMUX_Adc1Tcomp1ToFlexioTrigger
enumerator kINPUTMUX_Adc1Tcomp2ToFlexioTrigger
enumerator kINPUTMUX_Adc1Tcomp3ToFlexioTrigger
enumerator kINPUTMUX_Cmp0OutToFlexioTrigger

enumerator kINPUTMUX_Cmp1OutToFlexioTrigger
enumerator kINPUTMUX_Pwm0A0Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm0A0Trig1ToFlexioTrigger
enumerator kINPUTMUX_Pwm0A1Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm0A1Trig1ToFlexioTrigger
enumerator kINPUTMUX_Pwm0A2Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm0A2Trig1ToFlexioTrigger
enumerator kINPUTMUX_Pwm0A3Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm0A3Trig1ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A0Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A0Trig1ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A1Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A1Trig1ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A2Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A2Trig1ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A3Trig0ToFlexioTrigger
enumerator kINPUTMUX_Pwm1A3Trig1ToFlexioTrigger
enumerator kINPUTMUX_EvtgOut0AToFlexioTrigger
enumerator kINPUTMUX_EvtgOut0BToFlexioTrigger
enumerator kINPUTMUX_EvtgOut1AToFlexioTrigger
enumerator kINPUTMUX_EvtgOut1BToFlexioTrigger
enumerator kINPUTMUX_EvtgOut2AToFlexioTrigger
enumerator kINPUTMUX_EvtgOut2BToFlexioTrigger
enumerator kINPUTMUX_EvtgOut3AToFlexioTrigger
enumerator kINPUTMUX_EvtgOut3BToFlexioTrigger
enumerator kINPUTMUX_TrigIn0ToFlexioTrigger
enumerator kINPUTMUX_TrigIn1ToFlexioTrigger
enumerator kINPUTMUX_TrigIn2ToFlexioTrigger
enumerator kINPUTMUX_TrigIn3ToFlexioTrigger
enumerator kINPUTMUX_TrigIn4ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm0Trig0ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm0Trig1ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm0Trig2ToFlexioTrigger

enumerator kINPUTMUX_Lpflexcomm1Trig0ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm1Trig1ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm1Trig2ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm2Trig0ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm2Trig1ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm2Trig2ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm3Trig0ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm3Trig1ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm3Trig2ToFlexioTrigger
enumerator kINPUTMUX_Lpflexcomm3Trig3ToFlexioTrigger
enumerator kINPUTMUX_WuuToFlexioTrigger

enum _inputmux_signal_t

INPUTMUX signal enable/disable type.

Values:

enumerator kINPUTMUX_PinInt0ToDma0Ch3Ena
DMA0 REQ ENABLE0 signal.
enumerator kINPUTMUX_PinInt1ToDma0Ch4Ena
enumerator kINPUTMUX_PinInt2ToDma0Ch5Ena
enumerator kINPUTMUX_PinInt3ToDma0Ch6Ena
enumerator kINPUTMUX_Ctimer0M0ToDma0Ch7Ena
enumerator kINPUTMUX_Ctimer0M1ToDma0Ch8Ena
enumerator kINPUTMUX_Ctimer1M0ToDma0Ch9Ena
enumerator kINPUTMUX_Ctimer1M1ToDma0Ch10Ena
enumerator kINPUTMUX_Ctimer2M0ToDma0Ch11Ena
enumerator kINPUTMUX_Ctimer2M1ToDma0Ch12Ena
enumerator kINPUTMUX_Ctimer3M0ToDma0Ch13Ena
enumerator kINPUTMUX_Ctimer3M1ToDma0Ch14Ena
enumerator kINPUTMUX_Ctimer4M0ToDma0Ch15Ena
enumerator kINPUTMUX_Ctimer4M1ToDma0Ch16Ena
enumerator kINPUTMUX_Wuu0ToDma0Ch17Ena
enumerator kINPUTMUX_Micfil0FifoRequestToDma0Ch18Ena
enumerator kINPUTMUX_Adc0FifoARequestToDma0Ch21Ena
enumerator kINPUTMUX_Adc0FifoBRequestToDma0Ch22Ena
enumerator kINPUTMUX_Adc1FifoARequestToDma0Ch23Ena

enumerator kINPUTMUX__Adc1FifoBRequestToDma0Ch24Ena
enumerator kINPUTMUX__HsCmp0DmaRequestToDma0Ch28Ena
enumerator kINPUTMUX__HsCmp1DmaRequestToDma0Ch29Ena
enumerator kINPUTMUX__Evtg0Out0AToDma0Ch31Ena
DMA0 REQ ENABLE1 signal.
enumerator kINPUTMUX__Evtg0Out0BToDma0Ch32Ena
enumerator kINPUTMUX__Evtg0Out1AToDma0Ch33Ena
enumerator kINPUTMUX__Evtg0Out1BToDma0Ch34Ena
enumerator kINPUTMUX__Evtg0Out2AToDma0Ch35Ena
enumerator kINPUTMUX__Evtg0Out2BToDma0Ch36Ena
enumerator kINPUTMUX__Evtg0Out3AToDma0Ch37Ena
enumerator kINPUTMUX__Evtg0Out3BToDma0Ch38Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt0ToDma0Ch39Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt1ToDma0Ch40Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt2ToDma0Ch41Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt3ToDma0Ch42Ena
enumerator kINPUTMUX__FlexPwm0ReqVal0ToDma0Ch43Ena
enumerator kINPUTMUX__FlexPwm0ReqVal1ToDma0Ch44Ena
enumerator kINPUTMUX__FlexPwm0ReqVal2ToDma0Ch45Ena
enumerator kINPUTMUX__FlexPwm0ReqVal3ToDma0Ch46Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt0ToDma0Ch47Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt1ToDma0Ch48Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt2ToDma0Ch49Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt3ToDma0Ch50Ena
enumerator kINPUTMUX__FlexPwm1ReqVal0ToDma0Ch51Ena
enumerator kINPUTMUX__FlexPwm1ReqVal1ToDma0Ch52Ena
enumerator kINPUTMUX__FlexPwm1ReqVal2ToDma0Ch53Ena
enumerator kINPUTMUX__FlexPwm1ReqVal3ToDma0Ch54Ena
enumerator kINPUTMUX__Lptmr0ToDma0Ch57Ena
enumerator kINPUTMUX__Lptmr1ToDma0Ch58Ena
enumerator kINPUTMUX__FlexCan0DmaRequestToDma0Ch59Ena
enumerator kINPUTMUX__FlexCan1DmaRequestToDma0Ch60Ena
enumerator kINPUTMUX__FlexIO0ShiftRegister0RequestToDma0Ch61Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister1RequestToDma0Ch62Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister2RequestToDma0Ch63Ena
DMA0 REQ ENABLE2 signal.

enumerator kINPUTMUX_FlexIO0ShiftRegister3RequestToDma0Ch64Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister4RequestToDma0Ch65Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister5RequestToDma0Ch66Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister6RequestToDma0Ch67Ena

enumerator kINPUTMUX_FlexIO0ShiftRegister7RequestToDma0Ch68Ena

enumerator kINPUTMUX_LpFlexcomm0RxToDma0Ch69Ena

enumerator kINPUTMUX_LpFlexcomm0TxToDma0Ch70Ena

enumerator kINPUTMUX_LpFlexcomm1RxToDma0Ch71Ena

enumerator kINPUTMUX_LpFlexcomm1TxToDma0Ch72Ena

enumerator kINPUTMUX_LpFlexcomm2RxToDma0Ch73Ena

enumerator kINPUTMUX_LpFlexcomm2TxToDma0Ch74Ena

enumerator kINPUTMUX_LpFlexcomm3RxToDma0Ch75Ena

enumerator kINPUTMUX_LpFlexcomm3TxToDma0Ch76Ena

enumerator kINPUTMUX_LpFlexcomm4RxToDma0Ch77Ena

enumerator kINPUTMUX_LpFlexcomm4TxToDma0Ch78Ena

enumerator kINPUTMUX_LpFlexcomm5RxToDma0Ch79Ena

enumerator kINPUTMUX_LpFlexcomm5TxToDma0Ch80Ena

enumerator kINPUTMUX_LpFlexcomm6RxToDma0Ch81Ena

enumerator kINPUTMUX_LpFlexcomm6TxToDma0Ch82Ena

enumerator kINPUTMUX_LpFlexcomm7RxToDma0Ch83Ena

enumerator kINPUTMUX_LpFlexcomm7TxToDma0Ch84Ena

enumerator kINPUTMUX_I3c0RxToDma0Ch95Ena

DMA0 REQ ENABLE3 signal.

enumerator kINPUTMUX_I3c0TxToDma0Ch96Ena

enumerator kINPUTMUX_I3c1RxToDma0Ch97Ena

enumerator kINPUTMUX_I3c1TxToDma0Ch98Ena

enumerator kINPUTMUX_Sai0RxToDma0Ch99Ena

enumerator kINPUTMUX_Sai0TxToDma0Ch100Ena

enumerator kINPUTMUX_Sai1RxToDma0Ch101Ena

enumerator kINPUTMUX_Sai1TxToDma0Ch102Ena

enumerator kINPUTMUX_Gpio0PinEventRequest0ToDma0Ch108Ena

enumerator kINPUTMUX__Gpio0PinEventRequest1ToDma0Ch109Ena
enumerator kINPUTMUX__Gpio1PinEventRequest0ToDma0Ch110Ena
enumerator kINPUTMUX__Gpio1PinEventRequest1ToDma0Ch111Ena
enumerator kINPUTMUX__Gpio2PinEventRequest0ToDma0Ch112Ena
enumerator kINPUTMUX__Gpio2PinEventRequest1ToDma0Ch113Ena
enumerator kINPUTMUX__Gpio3PinEventRequest0ToDma0Ch114Ena
enumerator kINPUTMUX__Gpio3PinEventRequest1ToDma0Ch115Ena
enumerator kINPUTMUX__Gpio4PinEventRequest0ToDma0Ch116Ena
enumerator kINPUTMUX__Gpio4PinEventRequest1ToDma0Ch117Ena
enumerator kINPUTMUX__Gpio5PinEventRequest0ToDma0Ch118Ena
enumerator kINPUTMUX__Gpio5PinEventRequest1ToDma0Ch119Ena
DMA1 REQ ENABLE0 signal.

enumerator kINPUTMUX__PinInt0ToDma1Ch3Ena
enumerator kINPUTMUX__PinInt1ToDma1Ch4Ena
enumerator kINPUTMUX__PinInt2ToDma1Ch5Ena
enumerator kINPUTMUX__PinInt3ToDma1Ch6Ena
enumerator kINPUTMUX__Ctimer0M0ToDma1Ch7Ena
enumerator kINPUTMUX__Ctimer0M1ToDma1Ch8Ena
enumerator kINPUTMUX__Ctimer1M0ToDma1Ch9Ena
enumerator kINPUTMUX__Ctimer1M1ToDma1Ch10Ena
enumerator kINPUTMUX__Ctimer2M0ToDma1Ch11Ena
enumerator kINPUTMUX__Ctimer2M1ToDma1Ch12Ena
enumerator kINPUTMUX__Ctimer3M0ToDma1Ch13Ena
enumerator kINPUTMUX__Ctimer3M1ToDma1Ch14Ena
enumerator kINPUTMUX__Ctimer4M0ToDma1Ch15Ena
enumerator kINPUTMUX__Ctimer4M1ToDma1Ch16Ena
enumerator kINPUTMUX__Wuu0ToDma1Ch17Ena
enumerator kINPUTMUX__Micfil0FifoRequestToDma1Ch18Ena
enumerator kINPUTMUX__Adc0FifoARequestToDma1Ch21Ena
enumerator kINPUTMUX__Adc0FifoBRequestToDma1Ch22Ena
enumerator kINPUTMUX__Adc1FifoARequestToDma1Ch23Ena
enumerator kINPUTMUX__Adc1FifoBRequestToDma1Ch24Ena
enumerator kINPUTMUX__HsCmp0DmaRequestToDma1Ch28Ena

enumerator kINPUTMUX__HsCmp1DmaRequestToDma1Ch29Ena
enumerator kINPUTMUX__Evtg0Out0AToDma1Ch31Ena
DMA1 REQ ENABLE1 signal.
enumerator kINPUTMUX__Evtg0Out0BToDma1Ch32Ena
enumerator kINPUTMUX__Evtg0Out1AToDma1Ch33Ena
enumerator kINPUTMUX__Evtg0Out1BToDma1Ch34Ena
enumerator kINPUTMUX__Evtg0Out2AToDma1Ch35Ena
enumerator kINPUTMUX__Evtg0Out2BToDma1Ch36Ena
enumerator kINPUTMUX__Evtg0Out3AToDma1Ch37Ena
enumerator kINPUTMUX__Evtg0Out3BToDma1Ch38Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt0ToDma1Ch39Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt1ToDma1Ch40Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt2ToDma1Ch41Ena
enumerator kINPUTMUX__FlexPwm0ReqCapt3ToDma1Ch42Ena
enumerator kINPUTMUX__FlexPwm0ReqVal0ToDma1Ch43Ena
enumerator kINPUTMUX__FlexPwm0ReqVal1ToDma1Ch44Ena
enumerator kINPUTMUX__FlexPwm0ReqVal2ToDma1Ch45Ena
enumerator kINPUTMUX__FlexPwm0ReqVal3ToDma1Ch46Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt0ToDma1Ch47Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt1ToDma1Ch48Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt2ToDma1Ch49Ena
enumerator kINPUTMUX__FlexPwm1ReqCapt3ToDma1Ch50Ena
enumerator kINPUTMUX__FlexPwm1ReqVal0ToDma1Ch51Ena
enumerator kINPUTMUX__FlexPwm1ReqVal1ToDma1Ch52Ena
enumerator kINPUTMUX__FlexPwm1ReqVal2ToDma1Ch53Ena
enumerator kINPUTMUX__FlexPwm1ReqVal3ToDma1Ch54Ena
enumerator kINPUTMUX__Lptmr0ToDma1Ch57Ena
enumerator kINPUTMUX__Lptmr1ToDma1Ch58Ena
enumerator kINPUTMUX__FlexCan0DmaRequestToDma1Ch59Ena
enumerator kINPUTMUX__FlexCan1DmaRequestToDma1Ch60Ena
enumerator kINPUTMUX__FlexIO0ShiftRegister0RequestToDma1Ch61Ena
enumerator kINPUTMUX__FlexIO0ShiftRegister1RequestToDma1Ch62Ena
enumerator kINPUTMUX__FlexIO0ShiftRegister2RequestToDma1Ch63Ena
DMA1 REQ ENABLE2 signal.

enumerator kINPUTMUX_FlexIO0ShiftRegister3RequestToDma1Ch64Ena
enumerator kINPUTMUX_FlexIO0ShiftRegister4RequestToDma1Ch65Ena
enumerator kINPUTMUX_FlexIO0ShiftRegister5RequestToDma1Ch66Ena
enumerator kINPUTMUX_FlexIO0ShiftRegister6RequestToDma1Ch67Ena
enumerator kINPUTMUX_FlexIO0ShiftRegister7RequestToDma1Ch68Ena
enumerator kINPUTMUX_LpFlexcomm0RxToDma1Ch69Ena
enumerator kINPUTMUX_LpFlexcomm0TxToDma1Ch70Ena
enumerator kINPUTMUX_LpFlexcomm1RxToDma1Ch71Ena
enumerator kINPUTMUX_LpFlexcomm1TxToDma1Ch72Ena
enumerator kINPUTMUX_LpFlexcomm2RxToDma1Ch73Ena
enumerator kINPUTMUX_LpFlexcomm2TxToDma1Ch74Ena
enumerator kINPUTMUX_LpFlexcomm3RxToDma1Ch75Ena
enumerator kINPUTMUX_LpFlexcomm3TxToDma1Ch76Ena
enumerator kINPUTMUX_LpFlexcomm4RxToDma1Ch77Ena
enumerator kINPUTMUX_LpFlexcomm4TxToDma1Ch78Ena
enumerator kINPUTMUX_LpFlexcomm5RxToDma1Ch79Ena
enumerator kINPUTMUX_LpFlexcomm5TxToDma1Ch80Ena
enumerator kINPUTMUX_LpFlexcomm6RxToDma1Ch81Ena
enumerator kINPUTMUX_LpFlexcomm6TxToDma1Ch82Ena
enumerator kINPUTMUX_LpFlexcomm7RxToDma1Ch83Ena
enumerator kINPUTMUX_LpFlexcomm7TxToDma1Ch84Ena
enumerator kINPUTMUX_ESpi0Ch0ToDma1Ch89Ena
enumerator kINPUTMUX_ESpi0Ch1ToDma1Ch90Ena
enumerator kINPUTMUX_I3c0RxToDma1Ch95Ena
DMA1 REQ ENABLE3 signal.
enumerator kINPUTMUX_I3c0TxToDma1Ch96Ena
enumerator kINPUTMUX_I3c1RxToDma1Ch97Ena
enumerator kINPUTMUX_I3c1TxToDma1Ch98Ena
enumerator kINPUTMUX_Sai0RxToDma1Ch99Ena
enumerator kINPUTMUX_Sai0TxToDma1Ch100Ena
enumerator kINPUTMUX_Sai1RxToDma1Ch101Ena
enumerator kINPUTMUX_Sai1TxToDma1Ch102Ena
enumerator kINPUTMUX_Gpio0PinEventRequest0ToDma1Ch108Ena

enumerator kINPUTMUX_Gpio0PinEventRequest1ToDma1Ch109Ena

enumerator kINPUTMUX_Gpio1PinEventRequest0ToDma1Ch110Ena

enumerator kINPUTMUX_Gpio1PinEventRequest1ToDma1Ch111Ena

enumerator kINPUTMUX_Gpio2PinEventRequest0ToDma1Ch112Ena

enumerator kINPUTMUX_Gpio2PinEventRequest1ToDma1Ch113Ena

enumerator kINPUTMUX_Gpio3PinEventRequest0ToDma1Ch114Ena

enumerator kINPUTMUX_Gpio3PinEventRequest1ToDma1Ch115Ena

enumerator kINPUTMUX_Gpio4PinEventRequest0ToDma1Ch116Ena

enumerator kINPUTMUX_Gpio4PinEventRequest1ToDma1Ch117Ena

enumerator kINPUTMUX_Gpio5PinEventRequest0ToDma1Ch118Ena

enumerator kINPUTMUX_Gpio5PinEventRequest1ToDma1Ch119Ena

typedef enum *inputmux_connection_t* inputmux_connection_t
INPUTMUX connections type.

typedef enum *inputmux_signal_t* inputmux_signal_t
INPUTMUX signal enable/disable type.

TIMER0CAPTSEL0
Periphinmux IDs.

TIMER0TRIGIN

TIMER1CAPTSEL0

TIMER1TRIGIN

TIMER2CAPTSEL0

TIMER2TRIGIN

SMARTDMAARCHB_INMUX0

PINTSEL0

FREQMEAS_REF_REG

FREQMEAS_TAR_REG

TIMER3CAPTSEL0

TIMER3TRIGIN

TIMER4CAPTSEL0

TIMER4TRIGIN

CMP0_TRIG_REG

ADC0_TRIG0

ADC1_TRIG0

QDC0_TRIG_REG

QDC0_HOME_REG
QDC0_INDEX_REG
QDC0_PHASEB_REG
QDC0_PHASEA_REG
QDC1_TRIG_REG
QDC1_HOME_REG
QDC1_INDEX_REG
QDC1_PHASEB_REG
QDC1_PHASEA_REG
FlexPWM0_SM0_EXTSYNC_REG
FlexPWM0_SM1_EXTSYNC_REG
FlexPWM0_SM2_EXTSYNC_REG
FlexPWM0_SM3_EXTSYNC_REG
FlexPWM0_SM0_EXT_A_REG
FlexPWM0_SM1_EXT_A_REG
FlexPWM0_SM2_EXT_A_REG
FlexPWM0_SM3_EXT_A_REG
FlexPWM0_EXTFORCE_REG
FlexPWM0_FAULT0_REG
FlexPWM0_FAULT1_REG
FlexPWM0_FAULT2_REG
FlexPWM0_FAULT3_REG
FlexPWM1_SM0_EXTSYNC_REG
FlexPWM1_SM1_EXTSYNC_REG
FlexPWM1_SM2_EXTSYNC_REG
FlexPWM1_SM3_EXTSYNC_REG
FlexPWM1_SM0_EXT_A_REG
FlexPWM1_SM1_EXT_A_REG
FlexPWM1_SM2_EXT_A_REG
FlexPWM1_SM3_EXT_A_REG
FlexPWM1_EXTFORCE_REG
FlexPWM1_FAULT0_REG
FlexPWM1_FAULT1_REG

FlexPWM1_FAULT2_REG

FlexPWM1_FAULT3_REG

PWM0_EXT_CLK_REG

PWM1_EXT_CLK_REG

EVTG_TRIG0_REG

EXT_TRIG0_REG

CMP1_TRIG_REG

FLEXCOMM0_TRIG_REG

FLEXCOMM1_TRIG_REG

FLEXCOMM2_TRIG_REG

FLEXCOMM3_TRIG_REG

FLEXCOMM4_TRIG_REG

FLEXCOMM5_TRIG_REG

FLEXCOMM6_TRIG_REG

FLEXCOMM7_TRIG_REG

FLEXIO_TRIG0_REG

DMA0_REQ_ENABLE0_REG

DMA0_REQ_ENABLE1_REG

DMA0_REQ_ENABLE2_REG

DMA0_REQ_ENABLE3_REG

DMA1_REQ_ENABLE0_REG

DMA1_REQ_ENABLE1_REG

DMA1_REQ_ENABLE2_REG

DMA1_REQ_ENABLE3_REG

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. –

2.39 INTM: Interrupt Monitor Driver

```
FSL_INTM_DRIVER_VERSION
```

INTM driver version.

```
enum __intm_monitor
```

Interrupt monitors.

Values:

```
enumerator kINTM_Monitor1
```

```
enumerator kINTM_Monitor2
```

enumerator kINTM_Monitor3

enumerator kINTM_Monitor4

typedef enum *_intm_monitor* intm_monitor_t

Interrupt monitors.

typedef struct *_intm_monitor_config* intm_monitor_config_t

INTM interrupt source configuration structure.

typedef struct *_intm_config* intm_config_t

INTM configuration structure.

void INTM_GetDefaultConfig(*intm_config_t* *config)

Fill in the INTM config struct with the default settings.

The default values are:

```
config[0].irqnumber = NotAvail_IRQn;
config[0].maxtimer = 1000U;
config[1].irqnumber = NotAvail_IRQn;
config[1].maxtimer = 1000U;
config[2].irqnumber = NotAvail_IRQn;
config[2].maxtimer = 1000U;
config[3].irqnumber = NotAvail_IRQn;
config[3].maxtimer = 1000U;
config->enable = false;
```

Parameters

- config – Pointer to user's INTM config structure.

void INTM_Init(INTM_Type *base, const *intm_config_t* *config)

Ungates the INTM clock and configures the peripheral for basic operation.

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

Parameters

- base – INTM peripheral base address
- config – Pointer to user's INTM config structure.

void INTM_Deinit(INTM_Type *base)

Disables the INTM module.

Parameters

- base – INTM peripheral base address

static inline void INTM_EnableCycleCount(INTM_Type *base, bool enable)

Enable the cycle count timer mode.

Monitor mode enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

Parameters

- base – INTM peripheral base address.
- enable – Enable the cycle count or not.

```
static inline void INTM_AckIrq(INTM_Type *base, IRQn_Type irq)
```

Interrupt Acknowledge.

Call this function in ISR to acknowledge interrupt.

Parameters

- base – INTM peripheral base address.
- irq – Handle interrupt number.

```
static inline void INTM_SetInterruptRequestNumber(INTM_Type *base, intm_monitor_t intms,  
                                                IRQn_Type irq)
```

Interrupt Request Select.

This function is used to set the interrupt request number to monitor or check.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.
- irq – Interrupt request number to monitor.

Returns

Select the interrupt request number to monitor.

```
static inline void INTM_SetMaxTime(INTM_Type *base, intm_monitor_t intms, uint32_t count)
```

Set the maximum count time.

This function is to set the maximum time from interrupt generation to confirmation.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.
- count – Timer maximum count.

```
static inline void INTM_ClearTimeCount(INTM_Type *base, intm_monitor_t intms)
```

Clear the timer period in units of count.

This function is used to clear the INTM_TIMERa register.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

```
static inline uint32_t INTM_GetTimeCount(INTM_Type *base, intm_monitor_t intms)
```

Gets the timer period in units of count.

This function is used to get the number of INTM clock cycles from interrupt request to confirmation interrupt processing. If this number exceeds the set maximum time, will be an error signal.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

```
static inline bool INTM_GetStatusFlags(INTM_Type *base, intm_monitor_t intms)
```

Interrupt monitor status.

This function indicates whether the INTM_TIMERa value has exceeded the INTM_LATENCYa value. If any interrupt source in INTM_TIMERa exceeds the programmed

delay value, the monitor state can be cleared by calling the `INTM_ClearTimeCount()` API to clear the corresponding `INTM_TIMERa` register.

Parameters

- `base` – INTM peripheral base address.
- `intms` – Programmable interrupt monitors.

Returns

Whether `INTM_TIMER` value has exceeded `INTM_LATENCY` value.
false: `INTM_TIMER` value has not exceeded the `INTM_LATENCY` value;
true: `INTM_TIMER` value has exceeded the `INTM_LATENCY` value.

`struct __intm_monitor_config`

`#include <fsl_intm.h>` INTM interrupt source configuration structure.

Public Members

`uint32_t maxtimer`

Set the maximum timer

`IRQn_Type irqnumber`

Select the interrupt request number to monitor.

`struct __intm_config`

`#include <fsl_intm.h>` INTM configuration structure.

Public Members

`bool enable`

Interrupt source monitor config. enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

2.40 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, calendaring, 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

2.41 Intrusion and Tamper Response Controller

2.42 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
```

2.43 Common Driver

```
FSL_COMMON_DRIVER_VERSION
    common driver version.
DEBUG_CONSOLE_DEVICE_TYPE_NONE
    No debug console.
DEBUG_CONSOLE_DEVICE_TYPE_UART
    Debug console based on UART.
DEBUG_CONSOLE_DEVICE_TYPE_LPUART
    Debug console based on LPUART.
DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
    Debug console based on LPSCI.
DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
    Debug console based on USBCDC.
DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
    Debug console based on FLEXCOMM.
```

DEBUG_CONSOLE_DEVICE_TYPE_IUART

Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART

Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART

Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO

Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI

Debug console based on QSCI.

MIN(*a*, *b*)

Computes the minimum of *a* and *b*.

MAX(*a*, *b*)

Computes the maximum of *a* and *b*.

UINT16_MAX

Max value of uint16_t type.

UINT32_MAX

Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(addr, val)

Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)

Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)

Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)

Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

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_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPi
Group number for ECSPi status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator `kStatusGroup_SDIO_SLV`
Group number for SDIO_SLV status codes.

enumerator `kStatusGroup_MECC`
Group number for MECC status codes.

enumerator `kStatusGroup_ENET_QOS`
Group number for ENET_QOS status codes.

enumerator `kStatusGroup_LOG`
Group number for LOG status codes.

enumerator `kStatusGroup_I3CBUS`
Group number for I3CBUS status codes.

enumerator `kStatusGroup_QSCI`
Group number for QSCI status codes.

enumerator `kStatusGroup_ELEMU`
Group number for ELEMU status codes.

enumerator `kStatusGroup_QUEUEDSPI`
Group number for QSPI status codes.

enumerator `kStatusGroup_POWER_MANAGER`
Group number for POWER_MANAGER status codes.

enumerator `kStatusGroup_IPED`
Group number for IPED status codes.

enumerator `kStatusGroup_ELS_PKC`
Group number for ELS PKC status codes.

enumerator `kStatusGroup_CSS_PKC`
Group number for CSS PKC status codes.

enumerator `kStatusGroup_HOSTIF`
Group number for HOSTIF status codes.

enumerator `kStatusGroup_CLIF`
Group number for CLIF status codes.

enumerator `kStatusGroup_BMA`
Group number for BMA status codes.

enumerator `kStatusGroup_NETC`
Group number for NETC status codes.

enumerator `kStatusGroup_ELE`
Group number for ELE status codes.

enumerator `kStatusGroup_GLIKEY`
Group number for GLIKEY status codes.

enumerator `kStatusGroup_AON_POWER`
Group number for AON_POWER status codes.

enumerator `kStatusGroup_AON_COMMON`
Group number for AON_COMMON status codes.

enumerator `kStatusGroup_ENDAT3`
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
Generic status for Success.

enumerator kStatus_Fail
Generic status for Fail.

enumerator kStatus_ReadOnly
Generic status for read only failure.

enumerator kStatus_OutOfRange
Generic status for out of range access.

enumerator kStatus_InvalidArgument
Generic status for invalid argument check.

enumerator kStatus_Timeout
Generic status for timeout.

enumerator kStatus_NoTransferInProgress
Generic status for no transfer in progress.

enumerator kStatus_Busy
Generic status for module is busy.

enumerator kStatus_NoData
Generic status for no data is found for the operation.

typedef int32_t status_t
Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values

The – allocated memory.

`void SDK_Free(void *ptr)`

Free memory.

Parameters

- `ptr` – The memory to be release.

`void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)`

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- `delayTime_us` – Delay time in unit of microsecond.
- `coreClock_Hz` – Core clock frequency with Hz.

`static inline status_t EnableIRQ(IRQn_Type interrupt)`

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- `interrupt` – The IRQ number.

Return values

- `kStatus_Success` – Interrupt enabled successfully
- `kStatus_Fail` – Failed to enable the interrupt

`static inline status_t DisableIRQ(IRQn_Type interrupt)`

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- `interrupt` – The IRQ number.

Return values

- `kStatus_Success` – Interrupt disabled successfully
- `kStatus_Fail` – Failed to disable the interrupt

`static inline status_t EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)`

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the EnableGlobalIRQ().

Returns

Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management

mechanism of primask. User is required to use the EnableGlobalIRQ() and DisableGlobalIRQ() in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ().

static inline bool __SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t newValue)

static inline uint32_t __SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

Unused	Major Version	Minor Version	Bug Fix
31 25 24	17 16	9 8	0

ARRAY_SIZE(x)

Computes the number of elements in an array.

UINT64_H(X)

Macro to get upper 32 bits of a 64-bit value

UINT64_L(X)

Macro to get lower 32 bits of a 64-bit value

SUPPRESS_FALL_THROUGH_WARNING()

For switch case code block, if case section ends without “break;” statement, there will be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, “SUPPRESS_FALL_THROUGH_WARNING();” need to be added at the end of each case section which misses “break;”statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK_HAS_DWT_CYCCNT

The chip supports DWT CYCCNT or not.

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.44 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.

enum `_lpadc_tune_value`

Define enumeration of tune value.

Values:

enumerator `kLPADC__TuneValue0`

Tune value 0.

enumerator `kLPADC__TuneValue1`

Tune value 1.

enumerator `kLPADC__TuneValue2`

Tune value 2.

enumerator `kLPADC__TuneValue3`

Tune value 3.

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 enum _lpadc_tune_value` `lpadc_tune_value_t`

Define enumeration of tune value.

`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;
```

(continues on next page)

(continued from previous page)

```
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;
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.

static inline void LPADC_RequestHighSpeedModeTrim(ADC_Type *base)

Request high speed mode trim calculation.

Parameters

- base – LPADC peripheral base address.

static inline int8_t LPADC_GetHighSpeedTrimValue(ADC_Type *base)

Get high speed mode trim value, the result is a 5-bit signed value between -16 and 15.

Note: The high speed mode trim value is used to minimize offset for high speed conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The calculated high speed mode trim value.

static inline void LPADC_SetHighSpeedTrimValue(ADC_Type *base, int8_t trimValue)

Set high speed mode trim value.

Note: If is possible to set the trim value manually, but it is recommended to use the LPADC_RequestHighSpeedModeTrim.

Parameters

- base – LPADC peripheral base address.
- trimValue – The trim value to be set.

static inline void LPADC_EnableHighSpeedConversionMode(ADC_Type *base, bool enable)

Enable/disable high speed conversion mode, if enabled conversions complete 2 or 3 ADCK cycles sooner compared to conversion cycle counts when high speed mode is disabled.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable high speed conversion mode:
 - **true** Enable high speed conversion mode;
 - **false** Disable high speed conversion mode.

static inline void LPADC_EnableExtraCycle(ADC_Type *base, bool enable)

Enable/disable an additional ADCK cycle to conversion.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable an additional ADCK cycle to conversion:
 - **true** Enable an additional ADCK cycle to conversion;
 - **false** Disable an additional ADCK cycle to conversion.

static inline void LPADC_SetTuneValue(ADC_Type *base, *lpadc_tune_value_t* tuneValue)

Set tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.
- tuneValue – The tune value to be set, please refer to *lpadc_tune_value_t*.

static inline *lpadc_tune_value_t* LPADC_GetTuneValue(ADC_Type *base)

Get tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The tune value, please refer to *lpadc_tune_value_t*.

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_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.

2.45 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: $F_{\text{target}} = (\text{RESULT} - 2) * F_{\text{reference}} / 2^{\text{REF\_SCALE}}$.

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 $2^{\text{ref\_scale}}$.

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 kFREQME__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 **ref-ClkScaleFactor** 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.

2.46 LPCMP: Low Power Analog Comparator Driver

```
void LPCMP_Init(LPCMP_Type *base, const lpcmp_config_t *config)
```

Initialize the LPCMP.

This function initializes the LPCMP module. The operations included are:

- Enabling the clock for LPCMP module.
- Configuring the comparator.
- Enabling the LPCMP module. Note: For some devices, multiple LPCMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the LPCMPs. Check the chip reference manual for the clock assignment of the LPCMP.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp_config_t” structure.

void LPCMP_Deinit(LPCMP_Type *base)

De-initializes the LPCMP module.

This function de-initializes the LPCMP module. The operations included are:

- Disabling the LPCMP module.
- Disabling the clock for LPCMP module.

This function disables the clock for the LPCMP. Note: For some devices, multiple LPCMP instance shares the same clock gate. In this case, before disabling the clock for the LPCMP, ensure that all the LPCMP instances are not used.

Parameters

- base – LPCMP peripheral base address.

void LPCMP_GetDefaultConfig(*lpcmp_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->enableCmpToDacLink = false;
config->useUnfilteredOutput = false;
config->enableInvertOutput = false;
config->hysteresisMode     = kLPCMP_HysteresisLevel0;
config->powerMode          = kLPCMP_LowSpeedPowerMode;
config->functionalSourceClock = kLPCMP_FunctionalClockSource0;
config->plusInputSrc       = kLPCMP_PlusInputSrcMux;
config->minusInputSrc      = kLPCMP_MinusInputSrcMux;
```

Parameters

- config – Pointer to “lpcmp_config_t” structure.

static inline void LPCMP_Enable(LPCMP_Type *base, bool enable)

Enable/Disable LPCMP module.

Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable the module, and “false” means disable the module.

void LPCMP_SetInputChannels(LPCMP_Type *base, uint32_t positiveChannel, uint32_t negativeChannel)

Select the input channels for LPCMP. This function determines which input is selected for the negative and positive mux.

Parameters

- base – LPCMP 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 LPCMP_EnableDMA(LPCMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events. Normally, the LPCMP 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 – LPCMP peripheral base address.
- enable – “true” means enable DMA support, and “false” means disable DMA support.

void LPCMP_SetFilterConfig(LPCMP_Type *base, const *lpcmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp_filter_config_t” structure.

void LPCMP_SetDACConfig(LPCMP_Type *base, const *lpcmp_dac_config_t* *config)

Configure the internal DAC module.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to “lpcmp_dac_config_t” structure. If config is “NULL”, disable internal DAC.

static inline void LPCMP_EnableInterrupts(LPCMP_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See “_lpcmp_interrupt_enable”.

static inline void LPCMP_DisableInterrupts(LPCMP_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See “_lpcmp_interrupt_enable”.

static inline uint32_t LPCMP_GetStatusFlags(LPCMP_Type *base)

Get the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.

Returns

Mask value for the asserted flags. See “_lpcmp_status_flags”.

static inline void LPCMP_ClearStatusFlags(LPCMP_Type *base, uint32_t mask)

Clear the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for the flags. See “_lpcmp_status_flags”.

static inline void LPCMP_EnableWindowMode(LPCMP_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 – LPCMP peripheral base address.
- enable – “true” means enable window mode, and “false” means disable window mode.

void LPCMP_SetWindowControl(LPCMP_Type *base, const *lpcmp_window_control_config_t* *config)

Configure the window control, users can use this API to implement operations on the window, such as inverting the window signal, setting the window closing event (only valid in windowing mode), and setting the COUTA signal after the window is closed (only valid in windowing mode).

Parameters

- base – LPCMP peripheral base address.
- config – Pointer “lpcmp_window_control_config_t” structure.

void LPCMP_SetRoundRobinConfig(LPCMP_Type *base, const *lpcmp_roundrobin_config_t* *config)

Configure the roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer “lpcmp_roundrobin_config_t” structure.

static inline void LPCMP_EnableRoundRobinMode(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable roundrobin mode, and “false” means disable roundrobin mode.

void LPCMP_SetRoundRobinInternalTimer(LPCMP_Type *base, uint32_t value)

brief Configure the roundrobin internal timer reload value.

param base LPCMP peripheral base address. param value RoundRobin internal timer reload value, allowed range: 0x0UL-0xFFFFFFFFUL.

static inline void LPCMP_EnableRoundRobinInternalTimer(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin internal timer, note that this function is only valid when using the internal trigger source.

Parameters

- base – LPCMP peripheral base address.
- enable – “true” means enable roundrobin internal timer, and “false” means disable roundrobin internal timer.

static inline void LPCMP_SetPreSetValue(LPCMP_Type *base, uint8_t mask)

Set preset value for all channels, users can set all channels' preset value through this API, for example, if the mask set to 0x03U means channel0 and channel2's preset value set to 1U and other channels' preset value set to 0U.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetComparisonResult(LPCMP_Type *base)

Get comparison results for all channels, users can get all channels' comparison results through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' comparison result.

static inline void LPCMP_ClearInputChangedFlags(LPCMP_Type *base, uint8_t mask)

Clear input changed flags for single channel or multiple channels, users can clear input changed flag of a single channel or multiple channels through this API, for example, if the mask set to 0x03U means clear channel0 and channel2's input changed flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetInputChangedFlags(LPCMP_Type *base)

Get input changed flags for all channels, Users can get all channels' input changed flags through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' changed flag.

FSL_LPCMP_DRIVER_VERSION

LPCMP driver version 2.3.2.

enum _lpcmp_status_flags

LPCMP status falgs mask.

Values:

enumerator kLPCMP_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kLPCMP_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kLPCMP_OutputRoundRobinEventFlag

Detects when any channel's last comparison result is different from the pre-set value in trigger mode.

enumerator kLPCMP_OutputAssertEventFlag

Return the current value of the analog comparator output. The flag does not support W1C.

enum _lpcmp_interrupt_enable

LPCMP interrupt enable/disable mask.

Values:

enumerator kLPCMP__OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kLPCMP__OutputFallingInterruptEnable

Comparator interrupt enable falling.

enumerator kLPCMP__RoundRobinInterruptEnable

Comparator round robin mode interrupt occurred when the comparison result changes for a given channel.

enum __lpcmp_hysteresis_mode

LPCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

Values:

enumerator kLPCMP__HysteresisLevel0

The hard block output has level 0 hysteresis internally.

enumerator kLPCMP__HysteresisLevel1

The hard block output has level 1 hysteresis internally.

enumerator kLPCMP__HysteresisLevel2

The hard block output has level 2 hysteresis internally.

enumerator kLPCMP__HysteresisLevel3

The hard block output has level 3 hysteresis internally.

enum __lpcmp_power_mode

LPCMP nano mode.

Values:

enumerator kLPCMP__LowSpeedPowerMode

Low speed comparison mode is selected.

enumerator kLPCMP__HighSpeedPowerMode

High speed comparison mode is selected.

enumerator kLPCMP__NanoPowerMode

Nano power comparator is enabled.

enum __lpcmp_dac_reference_voltage_source

Internal DAC reference voltage source.

Values:

enumerator kLPCMP__VrefSourceVin1

vrefh_int is selected as resistor ladder network supply reference Vin.

enumerator kLPCMP__VrefSourceVin2

vrefh_ext is selected as resistor ladder network supply reference Vin.

enum __lpcmp_functional_source_clock

LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

Values:

enumerator kLPCMP__FunctionalClockSource0

Select functional mode clock source0.

enumerator kLPCMP_FunctionalClockSource1

Select functional mode clock source1.

enumerator kLPCMP_FunctionalClockSource2

Select functional mode clock source2.

enumerator kLPCMP_FunctionalClockSource3

Select functional mode clock source3.

enum __lpcmp_couta_signal

Set the COUTA signal value when the window is closed.

Values:

enumerator kLPCMP_COUTASignalNoSet

NO set the COUTA signal value when the window is closed.

enumerator kLPCMP_COUTASignalLow

Set COUTA signal low(0) when the window is closed.

enumerator kLPCMP_COUTASignalHigh

Set COUTA signal high(1) when the window is closed.

enum __lpcmp_close_window_event

Set COUT event, which can close the active window in window mode.

Values:

enumerator kLPCMP_CloseWindowEventNoSet

No Set COUT event, which can close the active window in window mode.

enumerator kLPCMP_CloseWindowEventRisingEdge

Set rising edge COUT signal as COUT event.

enumerator kLPCMP_CloseWindowEventFallingEdge

Set falling edge COUT signal as COUT event.

enumerator kLPCMP_CloseWindowEventBothEdge

Set both rising and falling edge COUT signal as COUT event.

enum __lpcmp_roundrobin_fixedmuxport

LPCMP round robin mode fixed mux port.

Values:

enumerator kLPCMP_FixedPlusMuxPort

Fixed plus mux port.

enumerator kLPCMP_FixedMinusMuxPort

Fixed minus mux port.

enum __lpcmp_roundrobin_clock_source

LPCMP round robin mode clock source selection.

Note: In different devices, the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

Values:

enumerator kLPCMP_RoundRobinClockSource0

Select roundrobin mode clock source0.

enumerator kLPCMP_RoundRobinClockSource1

Select roundrobin mode clock source1.

enumerator kLPCMP_RoundRobinClockSource2

Select roundrobin mode clock source2.

enumerator kLPCMP_RoundRobinClockSource3

Select roundrobin mode clock source3.

enum _lpcmp_roundrobin_trigger_source

LPCMP round robin mode trigger source.

Values:

enumerator kLPCMP_TriggerSourceExternally

Select external trigger source.

enumerator kLPCMP_TriggerSourceInternally

Select internal trigger source.

typedef enum _lpcmp_hysteresis_mode lpcmp_hysteresis_mode_t

LPCMP hysteresis mode. See chip data sheet to get the actual hysteresis value with each level.

typedef enum _lpcmp_power_mode lpcmp_power_mode_t

LPCMP nano mode.

typedef enum _lpcmp_dac_reference_voltage_source lpcmp_dac_reference_voltage_source_t

Internal DAC reference voltage source.

typedef enum _lpcmp_functional_source_clock lpcmp_functional_source_clock_t

LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

typedef enum _lpcmp_couta_signal lpcmp_couta_signal_t

Set the COUTA signal value when the window is closed.

typedef enum _lpcmp_close_window_event lpcmp_close_window_event_t

Set COUT event, which can close the active window in window mode.

typedef enum _lpcmp_roundrobin_fixedmuxport lpcmp_roundrobin_fixedmuxport_t

LPCMP round robin mode fixed mux port.

typedef enum _lpcmp_roundrobin_clock_source lpcmp_roundrobin_clock_source_t

LPCMP round robin mode clock source selection.

Note: In different devices, the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

typedef enum _lpcmp_roundrobin_trigger_source lpcmp_roundrobin_trigger_source_t

LPCMP round robin mode trigger source.

typedef struct _lpcmp_filter_config lpcmp_filter_config_t

Configure the filter.

typedef struct _lpcmp_dac_config lpcmp_dac_config_t

configure the internal DAC.

typedef struct _lpcmp_config lpcmp_config_t

Configures the comparator.

typedef struct _lpcmp_window_control_config lpcmp_window_control_config_t

Configure the window mode control.

```
typedef struct _lpcmp_roundrobin_config lpcmp_roundrobin_config_t
```

Configure the round robin mode.

```
LPCMP_CCR1_COUTA_CFG_MASK
```

```
LPCMP_CCR1_COUTA_CFG_SHIFT
```

```
LPCMP_CCR1_COUTA_CFG(x)
```

```
LPCMP_CCR1_EVT_SEL_CFG_MASK
```

```
LPCMP_CCR1_EVT_SEL_CFG_SHIFT
```

```
LPCMP_CCR1_EVT_SEL_CFG(x)
```

```
struct _lpcmp_filter_config
```

#include <fsl_lpcmp.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 _lpcmp_dac_config
```

#include <fsl_lpcmp.h> configure the internal DAC.

Public Members

bool enableLowPowerMode

Decide whether to enable DAC low power mode.

lpcmp_dac_reference_voltage_source_t referenceVoltageSource

Internal DAC supply voltage reference source.

uint8_t DACValue

Value for the DAC Output Voltage. Different devices has different available range, for specific values, please refer to the reference manual.

```
struct _lpcmp_config
```

#include <fsl_lpcmp.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 invert the comparator output.

`lpcmp_hysteresis_mode_t` hysteresisMode

LPCMP hysteresis mode.

`lpcmp_power_mode_t` powerMode

LPCMP power mode.

`lpcmp_functional_source_clock_t` functionalSourceClock

Select LPCMP functional mode clock source.

`struct __lpcmp_window_control_config`

`#include <fsl_lpcmp.h>` Configure the window mode control.

Public Members

`bool enableInvertWindowSignal`

True: enable invert window signal, False: disable invert window signal.

`lpcmp_couta_signal_t` COUTASignal

Decide whether to define the COUTA signal value when the window is closed.

`lpcmp_close_window_event_t` closeWindowEvent

Decide whether to select COUT event signal edge defines a COUT event to close window.

`struct __lpcmp_roundrobin_config`

`#include <fsl_lpcmp.h>` Configure the round robin mode.

Public Members

`uint8_t initDelayModules`

Comparator and DAC initialization delay modulus, See Reference Manual and DataSheet for specific value.

`uint8_t sampleClockNumbers`

Specify the number of the round robin clock cycles(0~3) to wait after scanning the active channel before sampling the channel's comparison result.

`uint8_t channelSampleNumbers`

Specify the number of samples for one channel, note that channelSampleNumbers must not smaller than sampleTimeThreshhold.

`uint8_t sampleTimeThreshhold`

Specify that for one channel, when (sampleTimeThreshhold + 1) sample results are "1", the final result is "1", otherwise the final result is "0", note that the sampleTimeThreshhold must not be larger than channelSampleNumbers.

`lpcmp_roundrobin_clock_source_t` roundrobinClockSource

Decide which clock source to choose in round robin mode.

`lpcmp_roundrobin_trigger_source_t` roundrobinTriggerSource

Decide which trigger source to choose in round robin mode.

`lpcmp_roundrobin_fixedmuxport_t` fixedMuxPort

Decide which mux port to choose as fixed channel in round robin mode.

`uint8_t fixedChannel`

Indicate which channel of the fixed mux port is used in round robin mode.

uint8_t checkerChannelMask

Indicate which channel of the non-fixed mux port to check its voltage value in round robin mode, for example, if checkerChannelMask set to 0x11U means select channel 0 and channel 4 as checker channel.

2.47 LPFLEXCOMM: LPFLEXCOMM Driver

2.48 LPFLEXCOMM Driver

FSL_LP_FLEXCOMM_DRIVER_VERSION

FlexCOMM driver version.

enum LP_FLEXCOMM_PERIPH_T

LP_FLEXCOMM peripheral modes.

Values:

enumerator LP_FLEXCOMM_PERIPH_NONE

No peripheral

enumerator LP_FLEXCOMM_PERIPH_LPUART

LPUART peripheral

enumerator LP_FLEXCOMM_PERIPH_LPSPI

LPSPI Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2C

LPI2C Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2CAndLPUART

LPI2C and LPUART Peripheral

enum _lpflexcomm_interrupt_flag

LP_FLEXCOMM interrupt source flags.

Values:

enumerator kLPFLEXCOMM_I2cSlaveInterruptFlag

enumerator kLPFLEXCOMM_I2cMasterInterruptFlag

enumerator kLPFLEXCOMM_SpiInterruptFlag

enumerator kLPFLEXCOMM_UartRxInterruptFlag

enumerator kLPFLEXCOMM_UartTxInterruptFlag

enumerator kLPFLEXCOMM_AllInterruptFlag

typedef void (*lpflexcomm_irq_handler_t)(uint32_t instance, void *handle)

Typedef for interrupt handler.

IRQn_Type const kFlexcommIrqs[]

Array with IRQ number for each LP_FLEXCOMM module.

uint32_t LP_FLEXCOMM_GetInstance(void *base)

Returns instance number for LP_FLEXCOMM module with given base address.

uint32_t LP_FLEXCOMM_GetBaseAddress(uint32_t instance)

Returns for LP_FLEXCOMM base address.

uint32_t LP_FLEXCOMM_GetInterruptStatus(uint32_t instance)

brief Returns for LP_FLEXCOMM interrupt source, see `_lpflexcomm_interrupt_flag`.

status_t LP_FLEXCOMM_Init(uint32_t instance, LP_FLEXCOMM_PERIPH_T periph)

Initializes LP_FLEXCOMM and selects peripheral mode according to the second parameter.

void LP_FLEXCOMM_Deinit(uint32_t instance)

Deinitializes LP_FLEXCOMM.

void LP_FLEXCOMM_SetIRQHandler(uint32_t instance, lpflexcomm_irq_handler_t handler,
void *lpflexcommHandle, LP_FLEXCOMM_PERIPH_T
periph)

Sets IRQ handler for given LP_FLEXCOMM module. It is used by drivers register IRQ handler according to LP_FLEXCOMM mode.

2.49 LPI2C: Low Power Inter-Integrated Circuit Driver

FSL_LPI2C_DRIVER_VERSION

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy

The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle

The slave driver is idle.

enumerator kStatus_LPI2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail

DMA request failed.

enumerator kStatus_LPI2C_Timeout

Timeout polling status flags.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.50 LPI2C Master Driver

`void LPI2C_MasterGetDefaultConfig(lpi2c_master_config_t *masterConfig)`

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig         = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns = 0;
masterConfig->pinLowTimeout_ns  = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable = false;
masterConfig->hostRequest.source  = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `lpi2c_master_config_t`.

`void LPI2C_MasterInit(LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t sourceClock_Hz)`

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `LPI2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`void LPI2C_MasterDeinit(LPI2C_Type *base)`

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

`void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const lpi2c_data_match_config_t *matchConfig)`

Configures LPI2C master data match feature.

Parameters

- `base` – The LPI2C peripheral base address.
- `matchConfig` – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_master_flags`

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

`_lpi2c_master_flags`.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_MasterGetStatusFlags()`.

`static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Enables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Disables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)`

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)`

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

`static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)`

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.


```
static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)
```

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

```
static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)
```

Sets the watermarks for LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txWords` – Transmit FIFO watermark value in words. The `kLPI2C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO is equal or less than `txWords`. Writing a value equal or greater than the FIFO size is truncated.
- `rxWords` – Receive FIFO watermark value in words. The `kLPI2C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO is greater than `rxWords`. Writing a value equal or greater than the FIFO size is truncated.

```
static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)
```

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- `base` – The LPI2C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address,
lpi2c_direction_t dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns *kStatus_LPI2C_Nak*.

Parameters

- *base* – The LPI2C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.

Return values

- *kStatus_Success* – Data was sent successfully.
- *kStatus_LPI2C_Busy* – Another master is currently utilizing the bus.
- *kStatus_LPI2C_Nak* – The slave device sent a NAK in response to a byte.
- *kStatus_LPI2C_FifoError* – FIFO under run or over run.
- *kStatus_LPI2C_ArbitrationLost* – Arbitration lost error.
- *kStatus_LPI2C_PinLowTimeout* – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- *base* – The LPI2C peripheral base address.
- *rxBuff* – The pointer to the data to be transferred.
- *rxSize* – The length in bytes of the data to be transferred.

Return values

- *kStatus_Success* – Data was received successfully.
- *kStatus_LPI2C_Busy* – Another master is currently utilizing the bus.
- *kStatus_LPI2C_Nak* – The slave device sent a NAK in response to a byte.
- *kStatus_LPI2C_FifoError* – FIFO under run or overrun.
- *kStatus_LPI2C_ArbitrationLost* – Arbitration lost error.
- *kStatus_LPI2C_PinLowTimeout* – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- *base* – The LPI2C peripheral base address.

Return values

- *kStatus_Success* – The STOP signal was successfully sent on the bus and the transaction terminated.
- *kStatus_LPI2C_Busy* – Another master is currently utilizing the bus.
- *kStatus_LPI2C_Nak* – The slave device sent a NAK in response to a byte.

- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)
Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.

- `handle` – Pointer to the LPI2C master driver handle.
- `transfer` – The pointer to the transfer descriptor.

Return values

- `kStatus__Success` – The transaction was started successfully.
- `kStatus__LPI2C__Busy` – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, lpi2c_master_handle_t *handle,  
                                     size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to the LPI2C master driver handle.
- `count` – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

```
void LPI2C_MasterTransferAbort(LPI2C_Type *base, lpi2c_master_handle_t *handle)
```

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to the LPI2C master driver handle.

Return values

- `kStatus__Success` – A transaction was successfully aborted.
- `kStatus__LPI2C__Idle` – There is not a non-blocking transaction currently in progress.

```
void LPI2C_MasterTransferHandleIRQ(uint32_t instance, void *lpi2cMasterHandle)
```

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `instance` – The LPI2C instance.
- `lpi2cMasterHandle` – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags

Errors to check for.

enum _lpi2c_direction

Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write

Master transmit.

enumerator kLPI2C_Read

Master receive.

enum _lpi2c_master_pin_config

LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source

LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin

Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger

Select the input trigger as the host request input

enum _lpi2c_host_request_polarity

LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow

Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Values:

enumerator kLPI2C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag
Don't send a stop condition.

typedef enum _lpi2c_direction lpi2c_direction_t
Direction of master and slave transfers.

typedef enum _lpi2c_master_pin_config lpi2c_master_pin_config_t
LPI2C pin configuration.

typedef enum _lpi2c_host_request_source lpi2c_host_request_source_t
LPI2C master host request selection.

typedef enum _lpi2c_host_request_polarity lpi2c_host_request_polarity_t
LPI2C master host request pin polarity configuration.


```
typedef struct _lpi2c_master_config lpi2c_master_config_t
```

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_data_match_config_mode lpi2c_data_match_config_mode_t
```

LPI2C master data match configuration modes.

```
typedef struct _lpi2c_match_config lpi2c_data_match_config_t
```

LPI2C master data match configuration structure.

```
typedef struct _lpi2c_master_transfer lpi2c_master_transfer_t
```

```
typedef struct _lpi2c_master_handle lpi2c_master_handle_t
```

```
typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t
*handle, status_t completionStatus, void *userData)
```

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base

The LPI2C peripheral base address.

Param completionStatus

Either kStatus_Success or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(uint32_t instance, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

```
bool enableMaster
```

Whether to enable master mode.

```
bool enableDoze
```

Whether master is enabled in doze mode.

```
bool debugEnable
```

Enable transfers to continue when halted in debug mode.

```
bool ignoreAck
```

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest

Host request options.

struct *_lpi2c_match_config*

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32_t match0

Match value 0.

uint32_t match1

Match value 1.

struct *_lpi2c_master_transfer*

#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration *_lpi2c_master_transfer_flags* for available options. Set to 0 or *kLPI2C_TransferDefaultFlag* for normal transfers.

uint16_t slaveAddress

The 7-bit slave address.

lpi2c_direction_t direction

Either *kLPI2C_Read* or *kLPI2C_Write*.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

struct __lpi2c_master_handle

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint16_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

struct hostRequest

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.51 LPI2C Master DMA Driver

```
void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                  edma_handle_t *rxDmaHandle, edma_handle_t
                                  *txDmaHandle, lpi2c_master_edma_transfer_callback_t
                                  callback, void *userData)
```

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – **[out]** Pointer to the LPI2C master driver handle.
- *rxDmaHandle* – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

Return values

- *kStatus_Success* – The transaction was started successfully.
- *kStatus_LPI2C_Busy* – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                           *handle, size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *count* – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- *kStatus_Success* –
- *kStatus_NoTransferInProgress* – There is not a DMA transaction currently in progress.

```
status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                         *handle)
```

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

```
typedef struct _lpi2c_master_edma_handle lpi2c_master_edma_handle_t
```

```
typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)
```

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Param base

The LPI2C peripheral base address.

Param handle

Handle associated with the completed transfer.

Param completionStatus

Either kStatus_Success or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_master_edma_handle
```

```
#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

```
LPI2C_Type *base
```

LPI2C base pointer.

```
bool isBusy
```

Transfer state machine current state.

```
uint8_t nbytes
```

eDMA minor byte transfer count initially configured.

`uint16_t` commandBuffer[10]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

`lpi2c_master_transfer_t` transfer

Copy of the current transfer info.

`lpi2c_master_edma_transfer_callback_t` completionCallback

Callback function pointer.

`void *userData`

Application data passed to callback.

`edma_handle_t *rx`

Handle for receive DMA channel.

`edma_handle_t *tx`

Handle for transmit DMA channel.

`edma_tcd_t` tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.52 LPI2C Slave Driver

`void` LPI2C_SlaveGetDefaultConfig(`lpi2c_slave_config_t` *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave      = true;
slaveConfig->address0         = 0U;
slaveConfig->address1         = 0U;
slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable = true;
slaveConfig->filterEnable     = true;
slaveConfig->enableGeneralCall = false;
slaveConfig->sclStall.enableAck = false;
slaveConfig->sclStall.enableTx  = true;
slaveConfig->sclStall.enableRx  = true;
slaveConfig->sclStall.enableAddress = true;
slaveConfig->ignoreAck         = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns = 0;
slaveConfig->sclGlitchFilterWidth_ns = 0;
slaveConfig->dataValidDelay_ns   = 0;
slaveConfig->clockHoldTime_ns    = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to `lpi2c_slave_config_t`.

`void` LPI2C_SlaveInit(`LPI2C_Type` *base, const `lpi2c_slave_config_t` *slaveConfig, `uint32_t` sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use LPI2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

[_lpi2c_slave_flags](#)

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag

- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

See also:

`_lpi2c_slave_flags`.

Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_SlaveGetStatusFlags()`.

`static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Enables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)`

Disables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)`

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

`static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)`

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- `base` – The LPI2C peripheral base address.
- `enableAddressValid` – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.

- `enableRx` – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- `enableTx` – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

`static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)`

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

`static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)`

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

Parameters

- `base` – The LPI2C peripheral base address.
- `ackOrNack` – Pass true for an ACK or false for a NAK.

`static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)`

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – True will enable ACKSTALL, false will disable ACKSTALL.

`static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)`

Returns the slave address sent by the I2C master.

This function should only be called if the `kLPI2C_SlaveAddressValidFlag` is asserted.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

`status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)`

Performs a polling send transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.
- `actualTxSize` – [out]

Returns

Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns

Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle,
lpi2c_slave_transfer_callback_t callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle,
uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kLPI2C_SlaveTransmitEvent* and *kLPI2C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to *lpi2c_slave_handle_t* structure which stores the transfer state.

- `eventMask` – Bit mask formed by OR'ing together `lpi2c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kLPI2C_SlaveAllEvents` to enable all events.

Return values

- `kStatus__Success` – Slave transfers were successfully started.
- `kStatus__LPI2C_Busy` – Slave transfers have already been started on this handle.

`status_t` LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- `kStatus__Success` –
- `kStatus__NoTransferInProgress` –

`void` LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

Return values

- `kStatus__Success` –
- `kStatus__LPI2C_Idle` –

`void` LPI2C_SlaveTransferHandleIRQ(uint32_t instance, void *lpi2cSlaveHandle)

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

- `instance` – The LPI2C instance.
- `lpi2cSlaveHandle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag

Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag

Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag

Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag

Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag

Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag

FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag

Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag

Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag

General call flag

enumerator kLPI2C_SlaveBusyFlag

Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag

Bus busy flag

enumerator kLPI2C_SlaveClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags

Errors to check for.

enum _lpi2c_slave_address_match

LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0

Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK. When this event is set, the driver will no longer decide to reply to ack/nack.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum _lpi2c_slave_address_match lpi2c_slave_address_match_t

LPI2C slave address match options.

```
typedef struct _lpi2c_slave_config lpi2c_slave_config_t
```

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_slave_transfer_event lpi2c_slave_transfer_event_t
```

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

```
typedef struct _lpi2c_slave_transfer lpi2c_slave_transfer_t
```

LPI2C slave transfer structure.

```
typedef struct _lpi2c_slave_handle lpi2c_slave_handle_t
```

```
typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, lpi2c_slave_transfer_t  
*transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base

Base address for the LPI2C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_slave_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Enable slave mode.

uint8_t address0

Slave's 7-bit address.

uint8_t address1

Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode
Address matching options.

bool filterDozeEnable
Enable digital glitch filter in doze mode.

bool filterEnable
Enable digital glitch filter.

bool enableGeneralCall
Enable general call address matching.

bool ignoreAck
Continue transfers after a NACK is detected.

bool enableReceivedAddressRead
Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns
Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns
Width in nanoseconds of the clock hold time.

struct *_lpi2c_slave_transfer*
#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event
Reason the callback is being invoked.

uint8_t receivedAddress
Matching address send by master.

uint8_t *data
Transfer buffer

size_t dataSize
Transfer size

status_t completionStatus
Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount
Number of bytes actually transferred since start or last repeated start.

struct *_lpi2c_slave_handle*
#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer
LPI2C slave transfer copy.

bool isBusy
Whether transfer is busy.

bool wasTransmit
Whether the last transfer was a transmit.

uint32_t eventMask
Mask of enabled events.

uint32_t transferredCount
Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck
Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDatSCLStall or enableAddressSCLStall.

bool enableTx
Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx
Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress
Enables SCL clock stretching when the address valid flag is asserted.

2.53 LPSPI: Low Power Serial Peripheral Interface

2.54 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure *lpspi_master_config_t*.

- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the *lpspi_master_config_t* structure to default values.

This API initializes the configuration structure for LPSPI_MasterInit(). The initialized structure can remain unchanged in LPSPI_MasterInit(), or can be modified before calling the LPSPI_MasterInit(). Example:

```
lpspi_master_config_t masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to *lpspi_master_config_t* structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure *lpspi_slave_config_t*.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the *lpspi_slave_config_t* structure to default values.

This API initializes the configuration structure for LPSPI_SlaveInit(). The initialized structure can remain unchanged in LPSPI_SlaveInit() or can be modified before calling the LPSPI_SlaveInit(). Example:

```
lpspi_slave_config_t slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to *lpspi_slave_config_t* structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)
Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)
Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)
Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)
Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)
Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)
Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)
Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type `_lpspi_flags`.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_interrupt_enable`.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum `_lpspi_dma_enable`.

static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

```
static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)
```

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

```
bool LPSPI_CheckTransferArgument(LPSPI_Type *base, lpspi_transfer_t *transfer, bool isEdma)
```

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

```
static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, lpspi_master_slave_mode_t mode)
```

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

```
static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, lpspi_which_pcs_t select)
```

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

```
static inline void LPSPI_SetPCSContinuous(LPSPI_Type *base, bool IsContinuous)
```

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinuous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum `_lpspi_pcs_polarity`.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame

size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

```
uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t  
                                srcClock_Hz, uint32_t *tcrPrescaleValue)
```

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

```
void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, lpspi_delay_type_t  
                                whichDelay)
```

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type *lpspi_delay_type_t*.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.

- `scaler` – The 8-bit delay value 0x00 to 0xFF (255).
- `whichDelay` – The desired delay to configure, must be of type `lpspi_delay_type_t`.

```
uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec,
                                   lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)
```

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type `lpspi_delay_type_t`.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPi module must be configured for master mode before configuring this. And note that the `delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler)`.

Parameters

- `base` – LPSPi peripheral address.
- `delayTimeInNanoSec` – The desired delay value in nano-seconds.
- `whichDelay` – The desired delay to configuration, which must be of type `lpspi_delay_type_t`.
- `srcClock_Hz` – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

```
static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)
```

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- `base` – LPSPi peripheral address.
- `data` – The data word to be sent.

```
static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)
```

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- `base` – LPSPi peripheral address.

Returns

The data read from the data buffer.

```
void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)
```

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

```
void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, lpspi_master_handle_t *handle,  
                                     lpspi_master_transfer_callback_t callback, void  
                                     *userData)
```

Initializes the LPSPI master handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to `lpspi_master_handle_t`.
- callback – DSPI callback.
- userData – callback function parameter.

```
status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, lpspi_transfer_t *transfer)
```

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of status_t.

```
status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, lpspi_master_handle_t *handle,  
                                         lpspi_transfer_t *transfer)
```

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of `status_t`.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(uint32_t instance, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- instance – LPSPI instance.
- handle – pointer to `lpspi_master_handle_t` structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to `lpspi_slave_handle_t`.
- callback – DSPI callback.
- userData – callback function parameter.

`status_t` LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of `status_t`.

`void` LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

`void` LPSPI_SlaveTransferHandleIRQ(uint32_t instance, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- instance – LPSPI instance index.
- handle – pointer to `lpspi_slave_handle_t` structure which stores the transfer state.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator `kStatus_LPSPI_Busy`

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag

Data Match flag

enumerator kLPSPI_ModuleBusyFlag

Module Busy flag

enumerator kLPSPI_AllStatusFlag

Used for clearing all w1c status flags

enum _lpspi_interrupt_enable

LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable

Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable

Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable

Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable

Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable

Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable

Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable

Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable

Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable

All above interrupts enable.

enum _lpspi_dma_enable

LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable

Receive data DMA enable

enum _lpspi_master_slave_mode

LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master

LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave

LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0

PCS[0]

enumerator kLPSPI_Pcs1

PCS[1]

enumerator kLPSPI_Pcs2

PCS[2]

enumerator kLPSPI_Pcs3

PCS[3]

enum _lpspi_pcs_polarity_config

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh

PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow

PCS Active Low (idles high)

enum `_lpspi_pcs_polarity`

LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator `kLPSPI_Pcs0ActiveLow`

Pcs0 Active Low (idles high).

enumerator `kLPSPI_Pcs1ActiveLow`

Pcs1 Active Low (idles high).

enumerator `kLPSPI_Pcs2ActiveLow`

Pcs2 Active Low (idles high).

enumerator `kLPSPI_Pcs3ActiveLow`

Pcs3 Active Low (idles high).

enumerator `kLPSPI_PcsAllActiveLow`

Pcs0 to Pcs5 Active Low (idles high).

enum `_lpspi_clock_polarity`

LPSPI clock polarity configuration.

Values:

enumerator `kLPSPI_ClockPolarityActiveHigh`

CPOL=0. Active-high LPSPI clock (idles low)

enumerator `kLPSPI_ClockPolarityActiveLow`

CPOL=1. Active-low LPSPI clock (idles high)

enum `_lpspi_clock_phase`

LPSPI clock phase configuration.

Values:

enumerator `kLPSPI_ClockPhaseFirstEdge`

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator `kLPSPI_ClockPhaseSecondEdge`

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum `_lpspi_shift_direction`

LPSPI data shifter direction options.

Values:

enumerator `kLPSPI_MsbFirst`

Data transfers start with most significant bit.

enumerator `kLPSPI_LsbFirst`

Data transfers start with least significant bit.

enum `_lpspi_host_request_select`

LPSPI Host Request select configuration.

Values:

enumerator `kLPSPI_HostReqExtPin`

Host Request is an ext pin.

enumerator kLPSPi_HostReqInternalTrigger

Host Request is an internal trigger.

enum _lpspi_match_config

LPSPi Match configuration options.

Values:

enumerator kLPSPi_MatchDisabled

LPSPi Match Disabled.

enumerator kLPSPi_1stWordEqualsM0orM1

LPSPi Match Enabled.

enumerator kLPSPi_AnyWordEqualsM0orM1

LPSPi Match Enabled.

enumerator kLPSPi_1stWordEqualsM0and2ndWordEqualsM1

LPSPi Match Enabled.

enumerator kLPSPi_AnyWordEqualsM0andNxtWordEqualsM1

LPSPi Match Enabled.

enumerator kLPSPi_1stWordAndM1EqualsM0andM1

LPSPi Match Enabled.

enumerator kLPSPi_AnyWordAndM1EqualsM0andM1

LPSPi Match Enabled.

enum _lpspi_pin_config

LPSPi pin (SDO and SDI) configuration.

Values:

enumerator kLPSPi_SdiInSdoOut

LPSPi SDI input, SDO output.

enumerator kLPSPi_SdiInSdiOut

LPSPi SDI input, SDI output.

enumerator kLPSPi_SdoInSdoOut

LPSPi SDO input, SDO output.

enumerator kLPSPi_SdoInSdiOut

LPSPi SDO input, SDI output.

enum _lpspi_data_out_config

LPSPi data output configuration.

Values:

enumerator kLpspiDataOutRetained

Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate

Data out is tristated when chip select is de-asserted

enum _lpspi_pcs_function_config

LPSPi cs function configuration.

Values:

enumerator kLPSPi_PcsAsCs

PCS pin select as cs function

enumerator kLPSPI_PcsAsData
PCS pin select as date function

enum _lpspi_transfer_width
LPSPI transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPI delay type selection.

Values:

enumerator kLPSPI_PcsToSck
PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
Delay between transfers.

enum _lpspi_transfer_config_flag_for_master
Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterWidth1
LPSPI master width shift macro, internal used LPSPI master transfer 1bit

enumerator kLPSPI_MasterWidth2
LPSPI master width shift macro, internal used LPSPI master transfer 2bit

enumerator kLPSPI_MasterWidth4
LPSPI master width shift macro, internal used LPSPI master transfer 4bit

enumerator kLPSPI_MasterPcsContinuous
Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap
Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

- i. If you set `bitPerFrame = 8`, no matter the `kLPSPi_MasterByteSwap` flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set `bitPerFrame = 16`: (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the `kLPSPi_MasterByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPi_MasterByteSwap` flag.
- iii. If you set `bitPerFrame = 32`: (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the `kLPSPi_MasterByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPi_MasterByteSwap` flag.

`enum _lpspi_transfer_config_flag_for_slave`

Use this enumeration for LPSPi slave transfer configFlags.

Values:

enumerator `kLPSPi_SlavePcs0`

LPSPi slave PCS shift macro, internal used. LPSPi slave transfer use PCS0 signal

enumerator `kLPSPi_SlavePcs1`

LPSPi slave PCS shift macro, internal used. LPSPi slave transfer use PCS1 signal

enumerator `kLPSPi_SlavePcs2`

LPSPi slave PCS shift macro, internal used. LPSPi slave transfer use PCS2 signal

enumerator `kLPSPi_SlavePcs3`

LPSPi slave PCS shift macro, internal used. LPSPi slave transfer use PCS3 signal

enumerator `kLPSPi_SlaveByteSwap`

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set `lpspi_shift_direction_t` to MSB).

- i. If you set `bitPerFrame = 8`, no matter the `kLPSPi_SlaveByteSwap` flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set `bitPerFrame = 16`: (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the `kLPSPi_SlaveByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPi_SlaveByteSwap` flag.
- iii. If you set `bitPerFrame = 32`: (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the `kLPSPi_SlaveByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPi_SlaveByteSwap` flag.

`enum _lpspi_transfer_state`

LPSPi transfer state, which is used for LPSPi transactional API state machine.

Values:

enumerator `kLPSPi_Idle`

Nothing in the transmitter/receiver.

enumerator `kLPSPi_Busy`

Transfer queue is not finished.

enumerator `kLPSPi_Error`

Transfer error.

`typedef enum _lpspi_master_slave_mode lpspi_master_slave_mode_t`

LPSPi master or slave mode configuration.

`typedef enum _lpspi_which_pcs_config lpspi_which_pcs_t`

LPSPi Peripheral Chip Select (PCS) configuration (which PCS to configure).

`typedef enum _lpspi_pcs_polarity_config lpspi_pcs_polarity_config_t`

LPSPi Peripheral Chip Select (PCS) Polarity configuration.


```
typedef enum _lpspi_clock_polarity lpspi_clock_polarity_t
    LPSPI clock polarity configuration.

typedef enum _lpspi_clock_phase lpspi_clock_phase_t
    LPSPI clock phase configuration.

typedef enum _lpspi_shift_direction lpspi_shift_direction_t
    LPSPI data shifter direction options.

typedef enum _lpspi_host_request_select lpspi_host_request_select_t
    LPSPI Host Request select configuration.

typedef enum _lpspi_match_config lpspi_match_config_t
    LPSPI Match configuration options.

typedef enum _lpspi_pin_config lpspi_pin_config_t
    LPSPI pin (SDO and SDI) configuration.

typedef enum _lpspi_data_out_config lpspi_data_out_config_t
    LPSPI data output configuration.

typedef enum _lpspi_pcs_function_config lpspi_pcs_function_config_t
    LPSPI cs function configuration.

typedef enum _lpspi_transfer_width lpspi_transfer_width_t
    LPSPI transfer width configuration.

typedef enum _lpspi_delay_type lpspi_delay_type_t
    LPSPI delay type selection.

typedef struct _lpspi_master_config lpspi_master_config_t
    LPSPI master configuration structure.

typedef struct _lpspi_slave_config lpspi_slave_config_t
    LPSPI slave configuration structure.

typedef struct _lpspi_master_handle lpspi_master_handle_t
    Forward declaration of the _lpspi_master_handle typedefs.

typedef struct _lpspi_slave_handle lpspi_slave_handle_t
    Forward declaration of the _lpspi_slave_handle typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPi_Type *base, lpspi_master_handle_t
*handle, status_t status, void *userData)
    Master completion callback function pointer type.

    Param base
        LPSPI peripheral address.

    Param handle
        Pointer to the handle for the LPSPI master.

    Param status
        Success or error code describing whether the transfer is completed.

    Param userData
        Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPi_Type *base, lpspi_slave_handle_t *handle,
status_t status, void *userData)
    Slave completion callback function pointer type.

    Param base
        LPSPI peripheral address.
```

Param handle

Pointer to the handle for the LPSPI slave.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

`typedef struct _lpspi_transfer lpspi_transfer_t`

LPSPI master/slave transfer structure.

`volatile uint8_t g_lpspiDummyData[]`

Global variable for dummy data value setting.

`LPSPI_DUMMY_DATA`

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

`SPI_RETRY_TIMES`

Retry times for waiting flag.

`LPSPI_MASTER_PCS_SHIFT`

LPSPI master PCS shift macro , internal used.

`LPSPI_MASTER_PCS_MASK`

LPSPI master PCS shift macro , internal used.

`LPSPI_MASTER_WIDTH_SHIFT`

LPSPI master width shift macro, internal used

`LPSPI_MASTER_WIDTH_MASK`

LPSPI master width shift mask, internal used

`LPSPI_SLAVE_PCS_SHIFT`

LPSPI slave PCS shift macro , internal used.

`LPSPI_SLAVE_PCS_MASK`

LPSPI slave PCS shift macro , internal used.

`struct _lpspi_master_config`

`#include <fsl_lpspi.h>` LPSPI master configuration structure.

Public Members

`uint32_t baudRate`

Baud Rate for LPSPI.

`uint32_t bitsPerFrame`

Bits per frame, minimum 8, maximum 4096.

`lpspi_clock_polarity_t cpol`

Clock polarity.

`lpspi_clock_phase_t cpha`

Clock phase.

`lpspi_shift_direction_t direction`

MSB or LSB data shift direction.

`uint32_t pcsToSckDelayInNanoSec`

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`uint32_t lastSckToPcsDelayInNanoSec`

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`uint32_t betweenTransferDelayInNanoSec`

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`lpspi_which_pcs_t whichPcs`

Desired Peripheral Chip Select (PCS).

`lpspi_pcs_polarity_config_t pcsActiveHighOrLow`

Desired PCS active high or low

`lpspi_pin_config_t pinCfg`

Configures which pins are used for input and output data during single bit transfers.

`lpspi_pcs_function_config_t pcsFunc`

Configures cs pins function.

`lpspi_data_out_config_t dataOutConfig`

Configures if the output data is tristated between accesses (LPSPi_PCS is negated).

`bool enableInputDelay`

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

`struct __lpspi_slave_config`

`#include <fsl_lpspi.h>` LPSPi slave configuration structure.

Public Members

`uint32_t bitsPerFrame`

Bits per frame, minimum 8, maximum 4096.

`lpspi_clock_polarity_t cpol`

Clock polarity.

`lpspi_clock_phase_t cpha`

Clock phase.

`lpspi_shift_direction_t direction`

MSB or LSB data shift direction.

`lpspi_which_pcs_t whichPcs`

Desired Peripheral Chip Select (pcs)

`lpspi_pcs_polarity_config_t pcsActiveHighOrLow`

Desired PCS active high or low

`lpspi_pin_config_t pinCfg`

Configures which pins are used for input and output data during single bit transfers.

`lpspi_data_out_config_t dataOutConfig`

Configures if the output data is tristated between accesses (LPSPi_PCS is negated).

`struct __lpspi_transfer`

`#include <fsl_lpspi.h>` LPSPi master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize

Transfer bytes.

uint32_t configFlags

Transfer transfer configuration flags. Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

struct `_lpspi_master_handle`

#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame

Number of bytes in each frame

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`uint32_t txBuffIfNull`
Used if the txData is NULL.

`volatile uint8_t state`
LPSPi transfer state , `_lpspi_transfer_state`.

`lpspi_master_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

`struct _lpspi_slave_handle`
#include <fsl_lpspi.h> LPSPi slave transfer handle structure used for transactional API.

Public Members

`volatile bool isByteSwap`
A flag that whether should byte swap.

`volatile uint8_t fifoSize`
FIFO dataSize.

`volatile uint8_t rxWatermark`
Rx watermark.

`volatile uint8_t bytesEachWrite`
Bytes for each write TDR.

`volatile uint8_t bytesEachRead`
Bytes for each read RDR.

`const uint8_t *volatile txData`
Send buffer.

`uint8_t *volatile rxData`
Receive buffer.

`volatile size_t txRemainingByteCount`
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`volatile uint8_t state`
LPSPi transfer state , `_lpspi_transfer_state`.

`volatile uint32_t errorCount`
Error count for slave transfer.

`lpspi_slave_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

2.55 LPSPi eDMA Driver

`FSL_LPSPi_EDMA_DRIVER_VERSION`

LPSPi EDMA driver version.

`typedef struct _lpspi_master_edma_handle lpspi_master_edma_handle_t`
Forward declaration of the `_lpspi_master_edma_handle` typedefs.

`typedef struct _lpspi_slave_edma_handle lpspi_slave_edma_handle_t`
Forward declaration of the `_lpspi_slave_edma_handle` typedefs.

`typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPi_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)`

Completion callback function pointer type.

Param base

LPSPi peripheral base address.

Param handle

Pointer to the handle for the LPSPi master.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

`typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPi_Type *base, lpspi_slave_edma_handle_t *handle, status_t status, void *userData)`

Completion callback function pointer type.

Param base

LPSPi peripheral base address.

Param handle

Pointer to the handle for the LPSPi slave.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

`void LPSPi_MasterTransferCreateHandleEDMA(LPSPi_Type *base, lpspi_master_edma_handle_t *handle, lpspi_master_edma_transfer_callback_t callback, void *userData, edma_handle_t *edmaRxRegToRxDataHandle, edma_handle_t *edmaTxDataToTxRegHandle)`

Initializes the LPSPI master eDMA handle.

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

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for `edmaRxRegToRxDataHandle`.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – LPSPI handle pointer to `lpspi_master_edma_handle_t`.
- `callback` – LPSPI callback.
- `userData` – callback function parameter.
- `edmaRxRegToRxDataHandle` – `edmaRxRegToRxDataHandle` pointer to `edma_handle_t`.
- `edmaTxDataToTxRegHandle` – `edmaTxDataToTxRegHandle` pointer to `edma_handle_t`.

`status_t` LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `transfer` – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, `uint32_t` configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- `base` – LPSPI peripheral base address.
- `handle` – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- `configFlags` – transfer configuration flags. `_lpspi_transfer_config_flag_for_master`.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPi_Busy – The LPSPi device is busy.

Returns

Indicates whether LPSPi master transfer was successful or not.

status_t LPSPi_MasterTransferEDMALite(LPSPi_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPi master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPi_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPi peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.
- transfer – pointer to *lpspi_transfer_t* structure, config field is not used.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPi_Busy – The LPSPi device is busy.
- kStatus_InvalidArgument – The transfer structure is invalid.

Returns

Indicates whether LPSPi master transfer was successful or not.

void LPSPi_MasterTransferAbortEDMA(LPSPi_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPi master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPi peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.

status_t LPSPi_MasterTransferGetCountEDMA(LPSPi_Type *base, *lpspi_master_edma_handle_t* *handle, *size_t* *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPi peripheral base address.
- handle – pointer to *lpspi_master_edma_handle_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns

status of status_t.

```
void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t
                                         *handle, lpspi_slave_edma_transfer_callback_t
                                         callback, void *userData, edma_handle_t
                                         *edmaRxRegToRxDataHandle, edma_handle_t
                                         *edmaTxDataToTxRegHandle)
```

Initializes the LPSPI slave eDMA handle.

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

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_slave_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

```
status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle,
                                  lpspi_transfer_t *transfer)
```

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

```
void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)
LPSPI slave aborts a transfer which is using eDMA.
```

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPi peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.

`status_t` LPSPi_SlaveTransferGetCountEDMA(LPSPi_Type *base, *lpspi_slave_edma_handle_t* *handle, `size_t` *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPi peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns

status of `status_t`.

`struct __lpspi_master_edma_handle`

#include <fsl_lpspi_edma.h> LPSPi master eDMA transfer handle structure used for transactional API.

Public Members

`volatile bool` isPcsContinuous

Is PCS continuous in transfer.

`volatile bool` isByteSwap

A flag that whether should byte swap.

`uint32_t` totalByteCount

Number of transfer bytes

`uint8_t` nbytes

eDMA minor byte transfer count initially configured.

`uint32_t` txBuffIfNull

Used if there is not txData for DMA purpose.

`uint32_t` rxBuffIfNull

Used if there is not rxData for DMA purpose.

`uint32_t` transmitCommand

Used to write TCR for DMA purpose.

`volatile int8_t` oneFifoBlockRxWatermark

Used to change RXWATER in FCR for DMA purpose.

`volatile int8_t` lastBytesRxWatermark

Used to change RXWATER in FCR for DMA purpose.

`volatile uint8_t` state

LPSPi transfer state , `_lpspi_transfer_state`.

lpspi_master_edma_transfer_callback_t callback

Completion callback.

`void *`userData

Callback user data.

```
edma_handle_t *edmaRxRegToRxDataHandle
    edma_handle_t handle point used for RxReg to RxData buff
edma_handle_t *edmaTxDataToTxRegHandle
    edma_handle_t handle point used for TxData to TxReg buff
edma_tcd_t lpspiSoftwareTCD[6]
    SoftwareTCD, internal used
struct __lpspi_slave_edma_handle
#include <fsl_lpspi_edma.h> LPSPI slave eDMA transfer handle structure used for transac-
tional API.
```

Public Members

```
uint32_t totalByteCount
    Number of transfer bytes
uint8_t nbytes
    eDMA minor byte transfer count initially configured.
uint32_t txBuffIfNull
    Used if there is not txData for DMA purpose.
uint32_t rxBuffIfNull
    Used if there is not rxData for DMA purpose.
volatile int8_t oneFifoBlockRxWatermark
    Used to change RXWATER in FCR for DMA purpose.
volatile int8_t lastBytesRxWatermark
    Used to change RXWATER in FCR for DMA purpose.
volatile uint8_t state
    LPSPI transfer state.
lpspi_slave_edma_transfer_callback_t callback
    Completion callback.
void *userData
    Callback user data.
edma_handle_t *edmaRxRegToRxDataHandle
    edma_handle_t handle point used for RxReg to RxData buff
edma_handle_t *edmaTxDataToTxRegHandle
    edma_handle_t handle point used for TxData to TxReg
edma_tcd_t lpspiSoftwareTCD[5]
    SoftwareTCD, internal used
```

2.56 LPTMR: Low-Power Timer

```
void LPTMR_Init(LPTMR_Type *base, const lptmr_config_t *config)
    Ungates the LPTMR clock and configures the peripheral for a basic operation.
```

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

Parameters

- base – LPTMR peripheral base address
- config – A pointer to the LPTMR configuration structure.

void LPTMR_Deinit(LPTMR_Type *base)

Gates the LPTMR clock.

Parameters

- base – LPTMR peripheral base address

void LPTMR_GetDefaultConfig(*lptmr_config_t* *config)

Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

Parameters

- config – A pointer to the LPTMR configuration structure.

static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)

Enables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration `lptmr_interrupt_enable_t`

static inline void LPTMR_DisableInterrupts(LPTMR_Type *base, uint32_t mask)

Disables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration `lptmr_interrupt_enable_t`.

static inline uint32_t LPTMR_GetEnabledInterrupts(LPTMR_Type *base)

Gets the enabled LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `lptmr_interrupt_enable_t`

static inline uint32_t LPTMR_GetStatusFlags(LPTMR_Type *base)

Gets the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `lptmr_status_flags_t`

```
static inline void LPTMR_ClearStatusFlags(LPTMR_Type *base, uint32_t mask)
```

Clears the LPTMR status flags.

Parameters

- `base` – LPTMR peripheral base address
- `mask` – The status flags to clear. This is a logical OR of members of the enumeration `lptmr_status_flags_t`.

```
static inline void LPTMR_SetTimerPeriod(LPTMR_Type *base, uint32_t ticks)
```

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note:

- a. The TCF flag is set with the CNR equals the count provided here and then increments.
 - b. Call the utility macros provided in the `fsl_common.h` to convert to ticks.
-

Parameters

- `base` – LPTMR peripheral base address
- `ticks` – A timer period in units of ticks

```
static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)
```

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

- `base` – LPTMR peripheral base address

Returns

The current counter value in ticks

```
static inline void LPTMR_StartTimer(LPTMR_Type *base)
```

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

- `base` – LPTMR peripheral base address

```
static inline void LPTMR_StopTimer(LPTMR_Type *base)
```

Stops the timer.

This function stops the timer and resets the timer's counter register.

Parameters

- `base` – LPTMR peripheral base address

FSL_LPTMR_DRIVER_VERSION

Driver Version

enum __lptmr_pin_select

LPTMR pin selection used in pulse counter mode.

Values:

enumerator kLPTMR_PinSelectInput_0

Pulse counter input 0 is selected

enumerator kLPTMR_PinSelectInput_1

Pulse counter input 1 is selected

enumerator kLPTMR_PinSelectInput_2

Pulse counter input 2 is selected

enumerator kLPTMR_PinSelectInput_3

Pulse counter input 3 is selected

enum __lptmr_pin_polarity

LPTMR pin polarity used in pulse counter mode.

Values:

enumerator kLPTMR_PinPolarityActiveHigh

Pulse Counter input source is active-high

enumerator kLPTMR_PinPolarityActiveLow

Pulse Counter input source is active-low

enum __lptmr_timer_mode

LPTMR timer mode selection.

Values:

enumerator kLPTMR_TimerModeTimeCounter

Time Counter mode

enumerator kLPTMR_TimerModePulseCounter

Pulse Counter mode

enum __lptmr_prescaler_glitch_value

LPTMR prescaler/glitch filter values.

Values:

enumerator kLPTMR_Prescale_Glitch_0

Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR_Prescale_Glitch_1

Prescaler divide 4, glitch filter 2

enumerator kLPTMR_Prescale_Glitch_2

Prescaler divide 8, glitch filter 4

enumerator kLPTMR_Prescale_Glitch_3

Prescaler divide 16, glitch filter 8

enumerator kLPTMR_Prescale_Glitch_4

Prescaler divide 32, glitch filter 16

enumerator kLPTMR_Prescale_Glitch_5

Prescaler divide 64, glitch filter 32

enumerator kLPTMR_Prescale_Glitch_6
Prescaler divide 128, glitch filter 64

enumerator kLPTMR_Prescale_Glitch_7
Prescaler divide 256, glitch filter 128

enumerator kLPTMR_Prescale_Glitch_8
Prescaler divide 512, glitch filter 256

enumerator kLPTMR_Prescale_Glitch_9
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR_Prescale_Glitch_10
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR_Prescale_Glitch_11
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR_Prescale_Glitch_12
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR_Prescale_Glitch_13
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR_Prescale_Glitch_14
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR_Prescale_Glitch_15
Prescaler divide 65536, glitch filter 32768

enum _lptmr_prescaler_clock_select
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

Values:

enum _lptmr_interrupt_enable
List of the LPTMR interrupts.

Values:

enumerator kLPTMR_TimerInterruptEnable
Timer interrupt enable

enum _lptmr_status_flags
List of the LPTMR status flags.

Values:

enumerator kLPTMR_TimerCompareFlag
Timer compare flag

typedef enum _lptmr_pin_select lptmr_pin_select_t
LPTMR pin selection used in pulse counter mode.

typedef enum _lptmr_pin_polarity lptmr_pin_polarity_t
LPTMR pin polarity used in pulse counter mode.

typedef enum _lptmr_timer_mode lptmr_timer_mode_t
LPTMR timer mode selection.

typedef enum *_lptmr_prescaler_glitch_value* lptmr_prescaler_glitch_value_t
LPTMR prescaler/glitch filter values.

typedef enum *_lptmr_prescaler_clock_select* lptmr_prescaler_clock_select_t
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

typedef enum *_lptmr_interrupt_enable* lptmr_interrupt_enable_t
List of the LPTMR interrupts.

typedef enum *_lptmr_status_flags* lptmr_status_flags_t
List of the LPTMR status flags.

typedef struct *_lptmr_config* lptmr_config_t
LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the LPTMR_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

static inline void LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)
Enable or disable timer DMA request.

Parameters

- base – base LPTMR peripheral base address
- enable – Switcher of timer DMA feature. “true” means to enable, “false” means to disable.

struct *_lptmr_config*
#include <fsl_lptmr.h> LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the LPTMR_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode
Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect
LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity
LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning
True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler
True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource
LPTMR clock source

lptmr_prescaler_glitch_value_t value

Prescaler or glitch filter value

2.57 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.58 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

void LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

```
void LPUART_GetDefaultConfig(lpuart_config_t *config)
```

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: `lpuartConfig->baudRate_Bps = 115200U`; `lpuartConfig->parityMode = kLPUART_ParityDisabled`; `lpuartConfig->dataBitsCount = kLPUART_EightDataBits`; `lpuartConfig->isMsb = false`; `lpuartConfig->stopBitCount = kLPUART_OneStopBit`; `lpuartConfig->txFifoWatermark = 0`; `lpuartConfig->rxFifoWatermark = 1`; `lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit`; `lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1`; `lpuartConfig->enableTx = false`; `lpuartConfig->enableRx = false`;

Parameters

- `config` – Pointer to a configuration structure.

```
status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
```

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the `LPUART_Init`.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- `base` – LPUART peripheral base address.
- `baudRate_Bps` – LPUART baudrate to be set.
- `srcClock_Hz` – LPUART clock source frequency in HZ.

Return values

- `kStatus_LPUART_BaudrateNotSupport` – Baudrate is not supported in the current clock source.
- `kStatus_Success` – Set baudrate succeeded.

```
void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)
```

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- `base` – LPUART peripheral base address.
- `enable` – true to enable, false to disable.

```
static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)
```

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer; otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

```
static inline void LPUART__EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)
```

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

```
static inline void LPUART__SetRxFifoWatermark(LPUART_Type *base, uint8_t water)
```

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

```
static inline void LPUART__SetTxFifoWatermark(LPUART_Type *base, uint8_t water)
```

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

```
static inline void LPUART__TransferEnable16Bit(lpuart_handle_t *handle, bool enable)
```

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in *lpuart_handle_t*.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

```
uint32_t LPUART__GetStatusFlags(LPUART_Type *base)
```

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *_lpuart_flags*. To check for a specific status, compare the return value with enumerators in the *_lpuart_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART__GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the `_lpuart_flags`.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: `kLPUART_TxDataRegEmptyFlag`, `kLPUART_TransmissionCompleteFlag`, `kLPUART_RxDataRegFullFlag`, `kLPUART_RxActiveFlag`, `kLPUART_NoiseErrorInRxDataRegFlag`, `kLPUART_ParityErrorInRxDataRegFlag`, `kLPUART_TxFifoEmptyFlag`, `kLPUART_RxFifoEmptyFlag`
Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the `_lpuart_status_flag_t` to do the OR operation and get the mask.

Return values

- `kStatus_LPUART_FlagCannotClearManually` – The flag can't be cleared by this function but it is cleared automatically by hardware.
- `kStatus_Success` – Status in the mask are cleared.

Returns

0 succeed, others failed.

`void` LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the `_lpuart_interrupt_enable`. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of the enumeration `_uart_interrupt_enable`.

`void` LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpuart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of `_lpuart_interrupt_enable`.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_lpuart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in `_lpuart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in `_lpuart_interrupt_enable`.

static inline uint32_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

`void LPUART_SendAddress(LPUART_Type *base, uint8_t address)`

Transmit an address frame in 9-bit data mode.

Parameters

- `base` – LPUART peripheral base address.
- `address` – LPUART slave address.

`status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)`

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the data to write.
- `length` – Size of the data to write.

Return values

- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully wrote all data.

`status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)`

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the data to write.
- `length` – Size of the data to write.

Return values

- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully wrote all data.

`status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)`

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the buffer to store the received data.
- `length` – Size of the buffer.

Return values

- `kStatus_LPUART_RxHardwareOverrun` – Receiver overrun happened while receiving data.
- `kStatus_LPUART_NoiseError` – Noise error happened while receiving data.

- `kStatus_LPUART_FramingError` – Framing error happened while receiving data.
- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`status_t` LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- `length` – Size of the buffer.

Return values

- `kStatus_LPUART_RxHardwareOverrun` – Receiver overrun happened while receiving data.
- `kStatus_LPUART_NoiseError` – Noise error happened while receiving data.
- `kStatus_LPUART_FramingError` – Framing error happened while receiving data.
- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`void` LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

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

The LPUART driver supports the “background” receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

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


```
status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, lpuart_handle_t *handle,  
                                         lpuart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

```
void LPUART_TransferStartRingBuffer(LPUART_Type *base, lpuart_handle_t *handle, uint8_t  
                                     *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

```
void LPUART_TransferStopRingBuffer(LPUART_Type *base, lpuart_handle_t *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.

- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, lpuart_handle_t *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, lpuart_handle_t *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see `uart_transfer_t`.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- `kStatus_Success` – Successfully queue the transfer into the transmit queue.
- `kStatus_LPUART_RxBusy` – Previous receive request is not finished.
- `kStatus_InvalidArgument` – Invalid argument.

`void LPUART_TransferAbortReceive(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the `remainBytes` to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

`status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

`void LPUART_TransferHandleIRQ(uint32_t instance, void *irqHandle)`

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- instance – LPUART instance.
- irqHandle – LPUART handle pointer.

`void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)`

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy
TX busy

enumerator kStatus_LPUART_RxBusy
RX busy

enumerator kStatus_LPUART_TxIdle
LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle
LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge
TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge
RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually
Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven

Parity enabled, type even, bit setting: PE | PT = 10

enumerator kLPUART_ParityOdd

Parity enabled, type odd, bit setting: PE | PT = 11

enum _lpuart_data_bits

LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits

Eight data bit

enumerator kLPUART_SevenDataBits

Seven data bit

enum _lpuart_stop_bit_count

LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit

One stop bit

enumerator kLPUART_TwoStopBit

Two stop bits

enum _lpuart_transmit_cts_source

LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin

CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult

CTS resource is the match result.

enum _lpuart_transmit_cts_config

LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart

CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle

CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select

LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit

Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit

Start counting after a stop bit.

enum _lpuart_idle_config

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable

LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_CtsStateChangeInterruptEnable
Change of state on CTS_B pin. bit 0

enumerator kLPUART_DsrStateChangeInterruptEnable
Change of state on DSR_B pin. bit 1

enumerator kLPUART_RinStateChangeInterruptEnable
Change of state on RIN_B pin. bit 2

enumerator kLPUART_DcdStateChangeInterruptEnable
Change of state on DCD_B pin. bit 3

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxCounter0TimeoutInterruptEnable
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutInterruptEnable
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutInterruptEnable
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutInterruptEnable

Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2InterruptEnable

The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1InterruptEnable

The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_IdleLineInterruptEnable

Idle line. bit 20

enumerator kLPUART_RxDataRegFullInterruptEnable

Receiver data register full. bit 21

enumerator kLPUART_TransmissionCompleteInterruptEnable

Transmission complete. bit 22

enumerator kLPUART_TxDataRegEmptyInterruptEnable

Transmit data register empty. bit 23

enumerator kLPUART_ParityErrorInterruptEnable

Parity error flag. bit 24

enumerator kLPUART_FramingErrorInterruptEnable

Framing error flag. bit 25

enumerator kLPUART_NoiseErrorInterruptEnable

Noise error flag. bit 26

enumerator kLPUART_RxOverrunInterruptEnable

Receiver Overrun. bit 27

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags

LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_RxFifoUnderflowFlag

RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_CtsStateChangeFlag

Change of state on CTS_B pin. bit 2

enumerator kLPUART_DsrStateChangeFlag

Change of state on DSR_B pin. bit 3

enumerator kLPUART_RinStateChangeFlag

Change of state on RIN_B pin. bit 4

enumerator kLPUART_DcdStateChangeFlag
Change of state on DCD_B pin. bit 5

enumerator kLPUART_RxCounter0TimeoutFlag
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutFlag
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutFlag
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutFlag
Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_RxActiveEdgeFlag
Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

enum `_lpuart_timeout_condition`

LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

Values:

enumerator `kLPUART_TimeoutAfterCharacters`

Timeout occurs when the number of characters specified by `timeoutValue` are received.

enumerator `kLPUART_TimeoutAfterIdle`

Timeout occurs when rx/tx remains idle for `timeoutValue` of bit clocks after idle condition is detected.

enumerator `kLPUART_TimeoutAfterNext`

Timeout occurs when rx/tx remains idle for `timeoutValue` of bit clocks after next character is received/transmitted.

enumerator `kLPUART_TimeoutAfterIdleBeforeExtended`

Timeout occurs when tx/rx is idle for larger than `timeoutValue` of bit clocks and smaller than tx/rx extended timeout value.

typedef enum `_lpuart_parity_mode` `lpuart_parity_mode_t`

LPUART parity mode.

typedef enum `_lpuart_data_bits` `lpuart_data_bits_t`

LPUART data bits count.

typedef enum `_lpuart_stop_bit_count` `lpuart_stop_bit_count_t`

LPUART stop bit count.

typedef enum `_lpuart_transmit_cts_source` `lpuart_transmit_cts_source_t`

LPUART transmit CTS source.

typedef enum `_lpuart_transmit_cts_config` `lpuart_transmit_cts_config_t`

LPUART transmit CTS configure.

typedef enum `_lpuart_idle_type_select` `lpuart_idle_type_select_t`

LPUART idle flag type defines when the receiver starts counting.

typedef enum `_lpuart_idle_config` `lpuart_idle_config_t`

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef enum `_lpuart_timeout_condition` `lpuart_timeout_condition_t`

LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

typedef struct `_lpuart_timeout_counter_config` `lpuart_timeout_counter_config_t`

LPUART timeout counter configuration structure.

typedef struct `_lpuart_timeout_config` `lpuart_timeout_config_t`

LPUART timeout configuration structure.

typedef struct `_lpuart_config` `lpuart_config_t`

LPUART configuration structure.

typedef struct `_lpuart_transfer` `lpuart_transfer_t`

LPUART transfer structure.

typedef struct `_lpuart_handle` `lpuart_handle_t`

```
typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle,
status_t status, void *userData)
```

LPUART transfer callback function.

```
typedef void (*lpuart_irq_handler_t)(uint32_t instance, void *handle)
```

```
const IRQn_Type s_lpuartIRQ[]
```

```
UART_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _lpuart_timeout_counter_config
```

#include <fsl_lpuart.h> LPUART timeout counter configuration structure.

Public Members

```
bool enableCounter
```

Enable the timeout counter.

```
lpuart_timeout_condition_t timeoutCondition
```

Timeout condition.

```
uint16_t timeoutValue
```

Timeout value.

```
struct _lpuart_timeout_config
```

#include <fsl_lpuart.h> LPUART timeout configuration structure.

Public Members

```
uint16_t rxExtendedTimeoutValue
```

The number of bits since the last stop bit that is required for an idle condition to be detected. Enable this will disable rxIdleType and rxIdleConfig. Set to 0 to disable.

```
uint16_t txExtendedTimeoutValue
```

The transmitter idle time in number of bits (baud rate) whenever an idle character is queued through the transmit FIFO.

```
lpuart_timeout_counter_config_t rxCounter0
```

Rx counter 0 configuration.

```
lpuart_timeout_counter_config_t rxCounter1
```

Rx counter 1 configuration.

```
lpuart_timeout_counter_config_t txCounter0
```

Tx counter 0 configuration.

```
lpuart_timeout_counter_config_t txCounter1
```

Tx counter 1 configuration.

```
struct _lpuart_config
```

#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

```
uint32_t baudRate_Bps
```

LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
RX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

lpuart_timeout_config_t timeoutConfig
Timeout configuration.

bool enableSingleWire
Use TXD pin as the source for the receiver. When enabled the TXD pin should be configured as open drain.

uint8_t rtsDelay
Delay the negation of RTS by the configured number of bit clocks.

bool enableTx
Enable TX

bool enableRx
Enable RX

struct __lpuart_transfer
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __lpuart_handle
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

volatile size_t txDataSize

Size of the remaining data to send.

size_t txDataSizeAll

Size of the data to send out.

volatile size_t rxDataSize

Size of the remaining data to receive.

size_t rxDataSizeAll

Size of the data to receive.

size_t rxRingBufferSize

Size of the ring buffer.

volatile uint16_t rxRingBufferHead

Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail

Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback

Callback function.

void *userData

LPUART callback function parameter.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state.

bool isSevenDataBits

Seven data bits flag.

bool is16bitData

16bit data bits flag, only used for 9bit or 10bit data

union __unnamed86__

Public Members

uint8_t *data

The buffer of data to be transfer.

uint8_t *rxData

The buffer to receive data.

uint16_t *rxData16

The buffer to receive data.

const uint8_t *txData

The buffer of data to be sent.

const uint16_t *txData16

The buffer of data to be sent.

union __unnamed88__

Public Members

const uint8_t *volatile txData
Address of remaining data to send.

const uint16_t *volatile txData16
Address of remaining data to send.

union __unnamed90__

Public Members

uint8_t *volatile rxData
Address of remaining data to receive.

uint16_t *volatile rxData16
Address of remaining data to receive.

union __unnamed92__

Public Members

uint8_t *rxRingBuffer
Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16
Start address of the receiver ring buffer.

2.59 LPUART eDMA Driver

```
void LPUART__TransferCreateHandleEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                                     lpuart_edma_transfer_callback_t callback, void  
                                     *userData, edma_handle_t *txEdmaHandle,  
                                     edma_handle_t *rxEdmaHandle)
```

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

```
status_t LPUART__SendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                          lpuart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See `lpuart_transfer_t`.

Return values

- `kStatus_Success` – if succeed, others failed.
- `kStatus_LPUART_TxBusy` – Previous transfer on going.
- `kStatus_InvalidArgument` – Invalid argument.

`status_t` LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.
- xfer – LPUART eDMA transfer structure, see `lpuart_transfer_t`.

Return values

- `kStatus_Success` – if succeed, others fail.
- `kStatus_LPUART_RxBusy` – Previous transfer ongoing.
- `kStatus_InvalidArgument` – Invalid argument.

`void` LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`void` LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`status_t` LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, `uint32_t` *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

- `count` – Send bytes count.

Return values

- `kStatus_NoTransferInProgress` – No send in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`status_t` LPUART__TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- `base` – LPUART peripheral base address.
- `handle` – LPUART handle pointer.
- `count` – Receive bytes count.

Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter `count`;

`void` LPUART__TransferEdmaHandleIRQ(uint32_t instance, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- `instance` – LPUART peripheral index.
- `lpuartEdmaHandle` – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

`typedef struct _lpuart_edma_handle` `lpuart_edma_handle_t`

`typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`

LPUART transfer callback function.

`struct _lpuart_edma_handle`

`#include <fsl_lpuart_edma.h>` LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback

Callback function.

`void *userData`
LPUART callback function parameter.

`size_t rxDataSizeAll`
Size of the data to receive.

`size_t txDataSizeAll`
Size of the data to send out.

`edma_handle_t *txEdmaHandle`
The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`
The eDMA RX channel used.

`uint8_t nbytes`
eDMA minor byte transfer count initially configured.

`volatile uint8_t txState`
TX transfer state.

`volatile uint8_t rxState`
RX transfer state

2.60 MCX_CMC: Core Mode Controller Driver

`enum __cmc_power_mode_protection`
CMC power mode Protection enumeration.

Values:

`enumerator kCMC__AllowDeepSleepMode`
Allow Deep Sleep mode.

`enumerator kCMC__AllowPowerDownMode`
Allow Power Down mode.

`enumerator kCMC__AllowDeepPowerDownMode`
Allow Deep Power Down mode.

`enumerator kCMC__AllowAllLowPowerModes`
Allow Deep Sleep, Power Down, Deep Power Down modes.

`enum __cmc_wakeup_sources`
Wake up sources from the previous low power mode entry.

Note: `kCMC_WakeupFromUsbFs`, `kCMC_WakeupFromITRC`, `kCMC_WakeupFromCpu1` are not supported in MCXA family.

Values:

`enumerator kCMC__WakeupFromResetInterruptOrPowerDown`
Wakeup source is reset interrupt, or wake up from Deep Power Down.

`enumerator kCMC__WakeupFromDebugReuquest`
Wakeup source is debug request.

`enumerator kCMC__WakeupFromInterrupt`
Wakeup source is interrupt.

enumerator kCMC_WakeupFromDMAWakeup

Wakeup source is DMA Wakeup.

enumerator kCMC_WakeupFromWUURquest

Wakeup source is WUU request.

enumerator kCMC_WakeupFromUsbFs

Wakeup source is USBFS(USB0).

enumerator kCMC_WakeupFromITRC

Wakeup source is ITRC.

enumerator kCMC_WakeupFromCpu1

Wakeup source is CPU1.

enum _cmc_system_reset_interrupt_enable

System Reset Interrupt enable enumeration.

Values:

enumerator kCMC_PinResetInterruptEnable

Pin Reset interrupt enable.

enumerator kCMC_DAPResetInterruptEnable

DAP Reset interrupt enable.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetInterruptEnable

Low Power Acknowledge Timeout Reset interrupt enable.

enumerator kCMC_SoftwareResetInterruptEnable

Software Reset interrupt enable.

enumerator kCMC_LockupResetInterruptEnable

Lockup Reset interrupt enable.

enum _cmc_system_reset_interrupt_flag

CMC System Reset Interrupt Status flag.

Values:

enumerator kCMC_PinResetInterruptFlag

Pin Reset interrupt flag.

enumerator kCMC_DAPResetInterruptFlag

DAP Reset interrupt flag.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetFlag

Low Power Acknowledge Timeout Reset interrupt flag.

enumerator kCMC_SoftwareResetInterruptFlag

Software Reset interrupt flag.

enumerator kCMC_LockupResetInterruptFlag

Lock up Reset interrupt flag.

enum _cmc_system_sram_arrays

CMC System SRAM arrays low power mode enable enumeration.

Values:

enumerator kCMC_RAMX0

Used to control RAMX0.

enumerator kCMC_RAMX1
Used to control RAMX1.

enumerator kCMC_RAMX2
Used to control RAMX2.

enumerator kCMC_RAMB
Used to control RAMB.

enumerator kCMC_RAMC0
Used to control RAMC0.

enumerator kCMC_RAMC1
Used to control RAMC1.

enumerator kCMC_RAMD0
Used to control RAMD0.

enumerator kCMC_RAMD1
Used to control RAMD1.

enumerator kCMC_RAME0
Used to control RAME0.

enumerator kCMC_RAME1
Used to control RAME1.

enumerator kCMC_RAMF0
Used to control RAMF0.

enumerator kCMC_RAMF1
Used to control RAMF1.

enumerator kCMC_RAMG0_RAMG1
Used to control RAMG0 and RAMG1.

enumerator kCMC_RAMG2_RAMG3
Used to control RAMG2 and RAMG3.

enumerator kCMC_RAMH0_RAMH1
Used to control RAMH0 and RAMH1.

enumerator kCMC_LPCAC
Used to control LPCAC.

enumerator kCMC_DMA0_DMA1_PKC
Used to control DMA0, DMA1 and PKC.

enumerator kCMC_USB0
Used to control USB0.

enumerator kCMC_PQ
Used to control PQ.

enumerator kCMC_CAN0_CAN1_ENET_USB1
Used to control CAN0, CAN1, ENET, USB1.

enumerator kCMC_FlexSPI
Used to control FlexSPI.

enumerator kCMC_AllSramArrays
Mask of all System SRAM arrays.

enum _cmc_system_reset_sources

System reset sources enumeration.

Values:

enumerator kCMC_WakeUpReset

The reset caused by a wakeup from Power Down or Deep Power Down mode.

enumerator kCMC_PORReset

The reset caused by power on reset detection logic.

enumerator kCMC_WarmReset

The last reset source is a warm reset source.

enumerator kCMC_FatalReset

The last reset source is a fatal reset source.

enumerator kCMC_PinReset

The reset caused by the RESET_b pin.

enumerator kCMC_DAPReset

The reset caused by a reset request from the Debug Access port.

enumerator kCMC_ResetTimeout

The reset caused by a timeout or other error condition in the system reset generation.

enumerator kCMC_LowPowerAcknowledgeTimeoutReset

The reset caused by a timeout in low power mode entry logic.

enumerator kCMC_SCGReset

The reset caused by a loss of clock or loss of lock event in the SCG.

enumerator kCMC_SoftwareReset

The reset caused by a software reset request.

enumerator kCMC_LockUoReset

The reset caused by the ARM core indication of a LOCKUP event.

enumerator kCMC_JTAGSystemReset

The reset caused by a JTAG system reset request.

enum _cmc_core_clock_gate_status

Indicate the core clock was gated.

Values:

enumerator kCMC_CoreClockNotGated

Core clock not gated.

enumerator kCMC_CoreClockGated

Core clock was gated due to low power mode entry.

enum _cmc_clock_mode

CMC clock mode enumeration.

Values:

enumerator kCMC_GateNoneClock

No clock gating.

enumerator kCMC_GateCoreClock

Gate Core clock.

enumerator kCMC_GateCorePlatformClock

Gate Core clock and platform clock.

enumerator kCMC_GateAllSystemClocks

Gate all System clocks, without getting core entering into low power mode.

enumerator kCMC_GateAllSystemClocksEnterLowPowerMode

Gate all System clocks, with core entering into low power mode.

enum _cmc_low_power_mode

CMC power mode enumeration.

Values:

enumerator kCMC_ActiveOrSleepMode

Select Active/Sleep mode.

enumerator kCMC_DeepSleepMode

Select Deep Sleep mode when a core executes WFI or WFE instruction.

enumerator kCMC_PowerDownMode

Select Power Down mode when a core executes WFI or WFE instruction.

enumerator kCMC_DeepPowerDown

Select Deep Power Down mode when a core executes WFI or WFE instruction.

typedef enum _cmc_core_clock_gate_status cmc_core_clock_gate_status_t

Indicate the core clock was gated.

typedef enum _cmc_clock_mode cmc_clock_mode_t

CMC clock mode enumeration.

typedef enum _cmc_low_power_mode cmc_low_power_mode_t

CMC power mode enumeration.

typedef struct _cmc_reset_pin_config cmc_reset_pin_config_t

CMC reset pin configuration.

typedef struct _cmc_power_domain_config cmc_power_domain_config_t

power mode configuration for each power domain.

FSL_CMC_DRIVER_VERSION

CMC driver version 2.5.0.

void CMC_SetClockMode(CMC_Type *base, cmc_clock_mode_t mode)

Sets clock mode.

This function configs the amount of clock gating when the core asserts Sleeping due to WFI, WFE or SLEEPONEXIT.

Parameters

- base – CMC peripheral base address.
- mode – System clock mode.

static inline void CMC_LockClockModeSetting(CMC_Type *base)

Locks the clock mode setting.

After invoking this function, any clock mode setting will be blocked.

Parameters

- base – CMC peripheral base address.

```
static inline cmc_core_clock_gate_status_t CMC_GetCoreClockGatedStatus(CMC_Type *base)
```

Gets the core clock gated status.

This function get the status to indicate whether the core clock is gated. The core clock gated status can be cleared by software.

Parameters

- base – CMC peripheral base address.

Returns

The status to indicate whether the core clock is gated.

```
static inline void CMC_ClearCoreClockGatedStatus(CMC_Type *base)
```

Clears the core clock gated status.

This function clear clock status flag by software.

Parameters

- base – CMC peripheral base address.

```
static inline uint8_t CMC_GetWakeupSource(CMC_Type *base)
```

Gets the Wakeup Source.

This function gets the Wakeup sources from the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Wakeup sources from the previous low power mode entry. See `_cmc_wakeup_sources` for details.

```
static inline cmc_clock_mode_t CMC_GetClockMode(CMC_Type *base)
```

Gets the Clock mode.

This function gets the clock mode of the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Low Power status.

```
static inline uint32_t CMC_GetSystemResetStatus(CMC_Type *base)
```

Gets the System reset status.

This function returns the system reset status. Those status updates on every MAIN Warm Reset to indicate the type/source of the most recent reset.

Parameters

- base – CMC peripheral base address.

Returns

The most recent system reset status. See `_cmc_system_reset_sources` for details.

```
static inline uint32_t CMC_GetStickySystemResetStatus(CMC_Type *base)
```

Gets the sticky system reset status since the last WAKE Cold Reset.

This function gets all source of system reset that have generated a system reset since the last WAKE Cold Reset, and that have not been cleared by software.

Parameters

- base – CMC peripheral base address.

Returns

System reset status that have not been cleared by software. See `_cmc_system_reset_sources` for details.

static inline void CMC_ClearStickySystemResetStatus(CMC_Type *base, uint32_t mask)

Clears the sticky system reset status flags.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the sticky system reset status to be cleared.

static inline uint8_t CMC_GetResetCount(CMC_Type *base)

Gets the number of reset sequences completed since the last Cold Reset.

Parameters

- base – CMC peripheral base address.

Returns

The number of reset sequences.

void CMC_SetPowerModeProtection(CMC_Type *base, uint32_t allowedModes)

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes. This should be done before set the lowPower mode for each power doamin.

The allowed lowpower modes are passed as bit map. For example, to allow Sleep and DeepSleep, use `CMC_SetPowerModeProtection(CMC_base, kCMC_AllowSleepMode | kCMC_AllowDeepSleepMode)`. To allow all low power modes, use `CMC_SetPowerModeProtection(CMC_base, kCMC_AllowAllLowPowerModes)`.

Parameters

- base – CMC peripheral base address.
- allowedModes – Bitmaps of the allowed power modes. See `_cmc_power_mode_protection` for details.

static inline void CMC_LockPowerModeProtectionSetting(CMC_Type *base)

Locks the power mode protection.

This function locks the power mode protection. After invoking this function, any power mode protection setting will be ignored.

Parameters

- base – CMC peripheral base address.

static inline void CMC_SetGlobalPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Config the same lowPower mode for all power domain.

This function configures the same low power mode for MAIN power domian and WAKE power domain.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See `cmc_low_power_mode_t` for details.

static inline void CMC_SetMAINPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Configures entry into low power mode for the MAIN Power domain.

This function configures the low power mode for the MAIN power domain, when the core executes WFI/WFE instruction. The available lowPower modes are defined in the `cmc_low_power_mode_t`.

Parameters

- `base` – CMC peripheral base address.
- `lowPowerMode` – The desired lowPower mode. See `cmc_low_power_mode_t` for details.

```
static inline cmc_low_power_mode_t CMC_GetMAINPowerMode(CMC_Type *base)
```

Gets the power mode of the MAIN Power domain.

Parameters

- `base` – CMC peripheral base address.

Returns

The power mode of MAIN Power domain. See `cmc_low_power_mode_t` for details.

```
void CMC_ConfigResetPin(CMC_Type *base, const cmc_reset_pin_config_t *config)
```

Configure reset pin.

This function configures reset pin. When enabled, the low power filter is enabled in both Active and Low power modes, the reset filter is only enabled in Active mode. When both filters are enabled, they operate in series.

Parameters

- `base` – CMC peripheral base address.
- `config` – Pointer to the reset pin config structure.

```
static inline void CMC_EnableSystemResetInterrupt(CMC_Type *base, uint32_t mask)
```

Enable system reset interrupts.

This function enables the system reset interrupts. The assertion of non-fatal warm reset can be delayed for 258 cycles of the 32K_CLK clock while an enabled interrupt is generated. Then Software can perform a graceful shutdown or abort the non-fatal warm reset provided the pending reset source is cleared by resetting the reset source and then clearing the pending flag.

Parameters

- `base` – CMC peripheral base address.
- `mask` – System reset interrupts. See `_cmc_system_reset_interrupt_enable` for details.

```
static inline void CMC_DisableSystemResetInterrupt(CMC_Type *base, uint32_t mask)
```

Disable system reset interrupts.

This function disables the system reset interrupts.

Parameters

- `base` – CMC peripheral base address.
- `mask` – System reset interrupts. See `_cmc_system_reset_interrupt_enable` for details.

```
static inline uint32_t CMC_GetSystemResetInterruptFlags(CMC_Type *base)
```

Gets System Reset interrupt flags.

This function returns the System reset interrupt flags.

Parameters

- base – CMC peripheral base address.

Returns

System reset interrupt flags. See `_cmc_system_reset_interrupt_flag` for details.

```
static inline void CMC_ClearSystemResetInterruptFlags(CMC_Type *base, uint32_t mask)
```

Clears System Reset interrupt flags.

This function clears system reset interrupt flags. The pending reset source can be cleared by resetting the source of the reset and then clearing the pending flags.

Parameters

- base – CMC peripheral base address.
- mask – System Reset interrupt flags. See `_cmc_system_reset_interrupt_flag` for details.

```
static inline void CMC_EnableNonMaskablePinInterrupt(CMC_Type *base, bool enable)
```

Enable/Disable Non maskable Pin interrupt.

Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Non maskable pin interrupt. true - enable Non-maskable pin interrupt. false - disable Non-maskable pin interrupt.

```
static inline uint8_t CMC_GetISPMODEPinLogic(CMC_Type *base)
```

Gets the logic state of the ISPMODE_n pin.

This function returns the logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

Parameters

- base – CMC peripheral base address.

Returns

The logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

```
static inline void CMC_ClearISPMODEPinLogic(CMC_Type *base)
```

Clears ISPMODE_n pin state.

Parameters

- base – CMC peripheral base address.

```
static inline void CMC_ForceBootConfiguration(CMC_Type *base, bool assert)
```

Set the logic state of the BOOT_CONFIGn pin.

This function force the logic state of the Boot_Confign pin to assert on next system reset.

Parameters

- base – CMC peripheral base address.
- assert – Assert the corresponding pin or not. true - Assert corresponding pin on next system reset. false - No effect.


```
static inline uint32_t CMC_GetBootRomStatus(CMC_Type *base)
```

Gets the status information written by the BootROM.

Parameters

- base – CMC peripheral base address.

Returns

The status information written by the BootROM.

```
static inline void CMC_SetBootRomStatus(CMC_Type *base, uint32_t statValue)
```

Sets the bootROM status value.

Note: This function is useful when result of CMC_CheckBootRomRegisterWrittable() is true.

Parameters

- base – CMC peripheral base address.
- stat – The state value to set.

```
static inline bool CMC_CheckBootRomRegisterWrittable(CMC_Type *base)
```

Check if BootROM status and lock registers is writable.

Parameters

- base – CMC peripheral base address.

Returns

The result of whether BootROM status and lock register is writable.

- **true** BootROM status and lock registers are writable;
- **false** BootROM status and lock registers are not writable.

```
static inline void CMC_LockBootRomStatusWritten(CMC_Type *base)
```

After invoking this function, BootROM status and lock registers cannot be written.

Parameters

- base – CMC peripheral base address.

```
static inline void CMC_UnlockBootRomStatusWritten(CMC_Type *base)
```

After invoking this function, BootROM status and lock register can be written.s.

Parameters

- base –

```
void CMC_PowerOffSRAMAllMode(CMC_Type *base, uint32_t mask)
```

Power off the selected system SRAM always.

Note: This function power off the selected system SRAM always. The SRAM arrays should not be accessed while they are shut down. SRAM array contents are not retained if they are powered off.

Note: Once invoked, the previous settings will be overwritten.

Parameters

- base – CMC peripheral base address.

- mask – Bitmap of the SRAM arrays to be powered off all modes. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC_PowerOnSRAMAllMode(CMC_Type *base, uint32_t mask)

Power on SRAM during all mode.

Note: Once invoked, the previous settings will be overwritten.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be powered on all modes. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC_PowerOffSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)

Power off the selected system SRAM during low power modes only.

This function power off the selected system SRAM only during low power mode. SRAM array contents are not retained if they are power off.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be power off during low power mode only. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC_PowerOnSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)

Power on the selected system SRAM during low power modes only.

This function power on the selected system SRAM. The SRAM array contents are retained in low power modes.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be power on during low power mode only. See `_cmc_system_sram_arrays` for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC_ConfigFlashMode(CMC_Type *base, bool doze, bool disable)

Configs the low power mode of the on-chip flash memory.

This function configs the low power mode of the on-chip flash memory.

Parameters

- base – CMC peripheral base address.
- doze – true: Flash is disabled while core is sleeping false: No effect.
- disable – true: Flash memory is placed in low power state. false: No effect.

static inline void CMC_EnableDebugOperation(CMC_Type *base, bool enable)

Enables/Disables debug Operation when the core sleep.

This function configs what happens to debug when core sleeps.

Parameters

- base – CMC peripheral base address.

- `enable` – Enable or disable Debug when Core is sleeping. `true` - Debug remains enabled when the core is sleeping. `false` - Debug is disabled when the core is sleeping.

`void CMC_PreEnterLowPowerMode(void)`

Prepares to enter low power modes.

This function should be called before entering low power modes.

`void CMC_PostExitLowPowerMode(void)`

Recovers after wake up from low power modes.

This function should be called after wake up from low power modes. This function should be used with `CMC_PreEnterLowPowerMode()`

`void CMC_GlobalEnterLowPowerMode(CMC_Type *base, cmc_low_power_mode_t lowPowerMode)`

Configs the entry into the same low power mode for each power domains.

This function provides the feature to entry into the same low power mode for each power domains. Before invoking this function, please ensure the selected power mode have been allowed.

Parameters

- `base` – CMC peripheral base address.
- `lowPowerMode` – The low power mode to be entered. See `cmc_low_power_mode_t` for the details.

`void CMC_EnterLowPowerMode(CMC_Type *base, const cmc_power_domain_config_t *config)`

Configs the entry into different low power modes for each power domains.

This function provides the feature to entry into different low power modes for each power domains. Before invoking this function please ensure the selected modes are allowed.

Parameters

- `base` – CMC peripheral base address.
- `config` – Pointer to the `cmc_power_domain_config_t` structure.

`bool lowpowerFilterEnable`

Low Power Filter enable.

`bool resetFilterEnable`

Reset Filter enable.

`uint8_t resetFilterWidth`

Width of the Reset Filter.

`cmc_clock_mode_t clock_mode`

Clock mode for each power domain.

`cmc_low_power_mode_t main_domain`

The low power mode of the MAIN power domain.

`struct _cmc_reset_pin_config`

`#include <fsl_cmc.h>` CMC reset pin configuration.

`struct _cmc_power_domain_config`

`#include <fsl_cmc.h>` power mode configuration for each power domain.

2.61 MCX_SPC: System Power Control driver

uint8_t SPC_GetPeriphIOIsolationStatus(SPC_Type *base)

Gets Isolation status for each power domains.

This function gets the status which indicates whether certain peripheral and the IO pads are in a latched state as a result of having been in POWERDOWN mode.

Parameters

- base – SPC peripheral base address.

Returns

Current isolation status for each power domains. See `_spc_power_domains` for details.

static inline void SPC_ClearPeriphIOIsolationFlag(SPC_Type *base)

Clears peripherals and I/O pads isolation flags for each power domains.

This function clears peripherals and I/O pads isolation flags for each power domains. After recovering from the POWERDOWN mode, user must invoke this function to release the I/O pads and certain peripherals to their normal run mode state. Before invoking this function, user must restore chip configuration in particular pin configuration for enabled WUU wakeup pins.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_GetBusyStatusFlag(SPC_Type *base)

Gets SPC busy status flag.

This function gets SPC busy status flag. When SPC executing any type of power mode transition in ACTIVE mode or any of the SOC low power mode, the SPC busy status flag is set and this function returns true. When changing CORE LDO voltage level and DCDC voltage level in ACTIVE mode, the SPC busy status flag is set and this function return true.

Parameters

- base – SPC peripheral base address.

Returns

Ack busy flag. true - SPC is busy. false - SPC is not busy.

static inline bool SPC_CheckLowPowerRequest(SPC_Type *base)

Checks system low power request.

Note: Only when all power domains request low power mode entry, the result of this function is true. That means when all power domains request low power mode entry, the SPC regulators will be controlled by LP_CFG register.

Parameters

- base – SPC peripheral base address.

Returns

The system low power request check result.

- **true** All power domains have requested low power mode and SPC has entered a low power state and power mode configuration are based on the LP_CFG configuration register.
- **false** SPC in active mode and ACTIVE_CFG register control system power supply.

```
static inline void SPC_ClearLowPowerRequest(SPC_Type *base)
```

Clears system low power request, set SPC in active mode.

Parameters

- `base` – SPC peripheral base address.

```
static inline spc_power_domain_low_power_mode_t SPC_GetRequestedLowPowerMode(SPC_Type  
                                                                           *base)
```

Check the last low-power mode that the power domain requested.

Parameters

- `base` – SPC peripheral base address.

Returns

The last low-power mode that the power domain requested.

```
static inline bool SPC_CheckSwitchState(SPC_Type *base)
```

Checks whether the power switch is on.

Parameters

- `base` – SPC peripheral base address.

Return values

- `true` – The power switch is on.
- `false` – The power switch is off.

```
spc_power_domain_low_power_mode_t SPC_GetPowerDomainLowPowerMode(SPC_Type *base,  
                                                                    spc_power_domain_id_t  
                                                                    powerDomainId)
```

Gets selected power domain's requested low power mode.

Parameters

- `base` – SPC peripheral base address.
- `powerDomainId` – Power Domain Id, please refer to `spc_power_domain_id_t`.

Returns

The selected power domain's requested low power mode, please refer to `spc_power_domain_low_power_mode_t`.

```
static inline bool SPC_CheckPowerDomainLowPowerRequest(SPC_Type *base,  
                                                        spc_power_domain_id_t  
                                                        powerDomainId)
```

Checks power domain's low power request.

Parameters

- `base` – SPC peripheral base address.
- `powerDomainId` – Power Domain Id, please refer to `spc_power_domain_id_t`.

Returns

The result of power domain's low power request.

- **true** The selected power domain requests low power mode entry.
- **false** The selected power domain does not request low power mode entry.

```
static inline void SPC_ClearPowerDomainLowPowerRequestFlag(SPC_Type *base,  
                                                           spc_power_domain_id_t  
                                                           powerDomainId)
```

Clears selected power domain's low power request flag.

Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to `spc_power_domain_id_t`.

```
static inline void SPC_TrimSRAMLdoRefVoltage(SPC_Type *base, uint8_t trimValue)
```

Trims SRAM retention regulator reference voltage, trim step is 12 mV, range is around 0.48V to 0.85V.

Parameters

- base – SPC peripheral base address.
- trimValue – Reference voltage trim value.

```
static inline void SPC_EnableSRAMLdo(SPC_Type *base, bool enable)
```

Enables/disables SRAM retention LDO.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable SRAM LDO :
 - **true** Enable SRAM LDO;
 - **false** Disable SRAM LDO.

```
static inline void SPC_RetainSRAMArray(SPC_Type *base, uint8_t mask)
```

Parameters

- base – SPC peripheral base address.
- mask – The OR'ed value of SRAM Array.

```
static inline void SPC_UnRetainSRAMArray(SPC_Type *base, uint8_t mask)
```

Unretain SRAM array.

Parameters

- base – SPC peripheral base address.
- mask – The OR'ed value of SRAM Array.

```
void SPC_SetLowPowerRequestConfig(SPC_Type *base, const spc_lowpower_request_config_t  
                                *config)
```

Configs Low power request output pin.

This function config the low power request output pin

Parameters

- base – SPC peripheral base address.
- config – Pointer the `spc_lowpower_request_config_t` structure.

```
static inline void SPC_EnableIntegratedPowerSwitchManually(SPC_Type *base, bool enable)
```

Enables/disables the integrated power switch manually.

Parameters

- base – SPC peripheral base address.

- enable – Used to enable/disable the integrated power switch:
 - **true** Enable the integrated power switch;
 - **false** Disable the integrated power switch.

```
static inline void SPC__EnableIntegratedPowerSwitchAutomatically(SPC_Type *base, bool
                                                                sleepGate, bool
                                                                wakeupUngate)
```

Enables/disables the integrated power switch automatically.

To gate the integrated power switch when chip enter low power modes, and ungate the switch after wake-up from low power modes:

```
SPC__EnableIntegratedPowerSwitchAutomatically(SPC, true, true);
```

Parameters

- base – SPC peripheral base address.
- sleepGate – Enable the integrated power switch when chip enter low power modes:
 - **true** SPC asserts an output pin at low-power entry to power-gate the switch;
 - **false** SPC does not assert an output pin at low-power entry to power-gate the switch.
- wakeupUngate – Enables the switch after wake-up from low power modes:
 - **true** SPC asserts an output pin at low-power exit to power-ungate the switch;
 - **false** SPC does not assert an output pin at low-power exit to power-ungate the switch.

```
void SPC__ConfigVddCoreGlitchDetector(SPC_Type *base, const
                                      spc_vdd_core_glitch_detector_config_t *config)
```

Configures VDD Core Glitch detector, including ripple counter selection, timeout value and so on.

Parameters

- base – SPC peripheral base address.
- config – Pointer to the structure in type of `spc_vdd_core_glitch_detector_config_t`.

```
static inline bool SPC__CheckGlitchRippleCounterOutput(SPC_Type *base,
                                                         spc_vdd_core_glitch_ripple_counter_select_t
                                                         rippleCounter)
```

Checks selected 4-bit glitch ripple counter's output.

Parameters

- base – SPC peripheral base address.
- rippleCounter – The ripple counter to check, please refer to `spc_vdd_core_glitch_ripple_counter_select_t`.

Return values

- true – The selected ripple counter output is 1, will generate interrupt or reset based on settings.
- false – The selected ripple counter output is 0.

```
static inline void SPC_ClearGlitchRippleCounterOutput(SPC_Type *base,  
                                                    spc_vdd_core_glitch_ripple_counter_select_t  
                                                    rippleCounter)
```

Clears output of selected glitch ripple counter.

Parameters

- base – SPC peripheral base address.
- rippleCounter – The ripple counter to check, please refer to `spc_vdd_core_glitch_ripple_counter_select_t`.

```
static inline void SPC_LockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)
```

After invoking this function, writes to SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register are ignored.

Parameters

- base – SPC peripheral base address.

```
static inline void SPC_UnlockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)
```

After invoking this function, writes to SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register are allowed.

Parameters

- base – SPC peripheral base address.

```
static inline bool SPC_CheckVddCoreVoltageGlitchResetControlState(SPC_Type *base)
```

Checks if SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.

Parameters

- base – SPC peripheral base address.

Return values

- true – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.
- false – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is not writable.

```
void SPC_SetSRAMOperateVoltage(SPC_Type *base, const spc_sram_voltage_config_t *config)
```

Set SRAM operate voltage.

Parameters

- base – SPC peripheral base address.
- config – The pointer to `spc_sram_voltage_config_t`, specifies the configuration of sram voltage.

```
static inline spc_bandgap_mode_t SPC_GetActiveModeBandgapMode(SPC_Type *base)
```

Gets the Bandgap mode in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

Bandgap mode in the type of `spc_bandgap_mode_t` enumeration.

```
static inline uint32_t SPC_GetActiveModeVoltageDetectStatus(SPC_Type *base)
```

Gets all voltage detectors status in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

All voltage detectors status in Active mode.

status_t SPC_SetActiveModeBandgapModeConfig(SPC_Type *base, *spc_bandgap_mode_t* mode)
Config Bandgap mode in Active mode.

Note: To disable bandgap in Active mode:

- a. Disable all LVD's and HVD's in active mode;
 - b. Disable Glitch detect;
 - c. Configure LDO's and DCDC to low drive strength in active mode;
 - d. Invoke this function to disable bandgap in active mode; otherwise the error status will be reported.
-

Note: Some other system resources(such as PLL, CMP) require bandgap to be enabled, to disable bandgap please take care of other system resources.

Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

Return values

- kStatus_SPC_BandgapModeWrong – The Bandgap can not be disabled in active mode.
- kStatus_Success – Config Bandgap mode in Active power mode successful.

static inline void SPC_EnableActiveModeCMPBandgapBuffer(SPC_Type *base, bool enable)
Enables/Disable the CMP Bandgap Buffer in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference voltage to CMP. false - Disable Buffer Stored Reference voltage to CMP.

static inline void SPC_SetActiveModeVoltageTrimDelay(SPC_Type *base, uint16_t delay)
Sets the delay when the regulators change voltage level in Active mode.

Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles.

status_t SPC_SetActiveModeRegulatorsConfig(SPC_Type *base, const
spc_active_mode_regulators_config_t *config)
Config all settings of regulators in Active mode at a time.

Note: This function is used to overwrite all settings of regulators(including bandgap mode, regulators' drive strength and voltage level) in active mode at a time.

Note: Enable/disable LVDs/HVDs before invoking this function.

Note: This function will check input parameters based on hardware restrictions before setting registers, if input parameters do not satisfy hardware restrictions the specific error will be reported.

Note: Some hardware restrictions not covered, application should be aware of this and follow this hardware restrictions otherwise some unknown issue may occur:

- a. If Core LDO's drive strength are set to same value in both Active mode and low power mode, the voltage level should also set to same value.
 - b. When switching Core LDO's drive strength from low to normal, ensure the LDO_CORE high voltage level is set to same level that was set prior to switching to the LDO_CORE drive strength. Otherwise, if the LVDs are enabled, an unexpected LVD can occur.
-

Note: If this function can not satisfy some tricky settings, please invoke other APIs in low-level function group.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_active_mode_regulators_config_t` structure.

Return values

- `kStatus_Success` – Config regulators in Active power mode successful.
- `kStatus_SPC_BandgapModeWrong` – Based on input setting, bandgap can not be disabled.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Any of LVDs/HVDs kept enabled before invoking this function.
- `kStatus_SPC_SYSLDOOverDriveVoltageFail` – Fail to regulator to Over Drive Voltage due to System VDD HVD is not disabled.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Any of LVDs/HVDs kept enabled before invoking this function.
- `kStatus_SPC_CORELDOVoltageWrong` – Core LDO and System LDO do not have same voltage level.

`static inline void SPC_DisableActiveModeVddCoreGlitchDetect(SPC_Type *base, bool disable)`
Disables/Enables VDD Core Glitch Detect in Active mode.

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
 - **true** Disable VDD Core Low Voltage detect;
 - **false** Enable VDD Core Low Voltage detect.

static inline bool SPC_CheckActiveModeVddCoreGlitchDetectEnabled(SPC_Type *base)

Check if Glitch detect hardware is enabled in active mode.

Parameters

- base – SPC peripheral base address.

Returns

Indicate if Glitch detector is enabled.

static inline void SPC_EnableActiveModeAnalogModules(SPC_Type *base, uint32_t maskValue)

Enables analog modules in active mode.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to enable in active mode, should be the OR'ed value of `spc_analog_module_control`.

static inline void SPC_DisableActiveModeAnalogModules(SPC_Type *base, uint32_t maskValue)

Disables analog modules in active mode.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to disable in active mode, should be the OR'ed value of `spc_analog_module_control`.

static inline uint32_t SPC_GetActiveModeEnabledAnalogModules(SPC_Type *base)

Gets enabled analog modules that enabled in active mode.

Parameters

- base – SPC peripheral base address.

Returns

The mask of enabled analog modules that enabled in active mode.

static inline *spc_bandgap_mode_t* SPC_GetLowPowerModeBandgapMode(SPC_Type *base)

Gets the Bandgap mode in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns

Bandgap mode in the type of `spc_bandgap_mode_t` enumeration.

static inline uint32_t SPC_GetLowPowerModeVoltageDetectStatus(SPC_Type *base)

Gets the status of all voltage detectors in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns

The status of all voltage detectors in low power mode.

static inline void SPC_EnableLowPowerModeLowPowerIREF(SPC_Type *base, bool enable)

Enables/Disables Low Power IREF in low power modes.

This function enables/disables Low Power IREF. Low Power IREF can only get disabled in Deep power down mode. In other low power modes, the Low Power IREF is always enabled.

Parameters

- base – SPC peripheral base address.

- `enable` – Enable/Disable Low Power IREF. `true` - Enable Low Power IREF for Low Power modes. `false` - Disable Low Power IREF for Deep Power Down mode.

`status_t` SPC_SetLowPowerModeBandgapmodeConfig(`SPC_Type` *base, `spc_bandgap_mode_t` mode)

Configs Bandgap mode in Low Power mode.

Note: To disable Bandgap in Low-power mode:

- a. Disable all LVD's and HVD's in low power mode;
 - b. Disable Glitch detect in low power mode;
 - c. Configure LDO's and DCDC to low drive strength in low power mode;
 - d. Disable bandgap in low power mode; Otherwise, the error status will be reported.
-

Note: Some other system resources(such as PLL, CMP) require bandgap to be enabled, to disable bandgap please take care of other system resources.

Parameters

- `base` – SPC peripheral base address.
- `mode` – The Bandgap mode to be selected.

Return values

- `kStatus_SPC_BandgapModeWrong` – The bandgap mode setting in Low Power mode is wrong.
- `kStatus_Success` – Config Bandgap mode in Low Power mode successful.

`static inline void` SPC_EnableSRAMLdOLowPowerModeIREF(`SPC_Type` *base, `bool` enable)
Enables/disables SRAM_LDO deep power low power IREF.

Parameters

- `base` – SPC peripheral base address.
- `enable` – Used to enable/disable low power IREF :
 - **true:** Low Power IREF is enabled ;
 - **false:** Low Power IREF is disabled for power saving.

`static inline void` SPC_EnableLowPowerModeCMPBandgapBufferMode(`SPC_Type` *base, `bool` enable)

Enables/Disables CMP Bandgap Buffer.

This function gates CMP bandgap buffer. CMP bandgap buffer is automatically disabled and turned off in Deep Power Down mode.

Deprecated:

No longer used, please use SPC_EnableLowPowerModeCMPBandgapBuffer as instead.

Parameters

- `base` – SPC peripheral base address.

- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference Voltage to CMP. false - Disable Buffer Stored Reference Voltage to CMP.

static inline void SPC_EnableLowPowerModeCMPBandgapBuffer(SPC_Type *base, bool enable)
Enables/Disables CMP Bandgap Buffer.

This function gates CMP bandgap buffer. CMP bandgap buffer is automatically disabled and turned off in Deep Power Down mode.

Deprecated:

No longer used.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference Voltage to CMP. false - Disable Buffer Stored Reference Voltage to CMP.

static inline void SPC_EnableLowPowerModeCoreVDDInternalVoltageScaling(SPC_Type *base,
bool enable)

Enables/Disables CORE VDD IVS(Internal Voltage Scaling) in power down modes.

This function gates CORE VDD IVS. When enabled, the IVS regulator will scale the external input CORE VDD to a lower voltage level to reduce internal leakage. IVS is invalid in Sleep or Deep power down mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IVS. true - enable CORE VDD IVS in Power Down mode. false - disable CORE VDD IVS in Power Down mode.

static inline void SPC_SetLowPowerWakeUpDelay(SPC_Type *base, uint16_t delay)
Sets the delay when exit the low power modes.

Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles that the SPC waits on exit from low power modes.

status_t SPC_SetLowPowerModeRegulatorsConfig(SPC_Type *base, const
spc_lowpower_mode_regulators_config_t
*config)

Configs all settings of regulators in Low power mode at a time.

Note: This function is used to overwrite all settings of regulators(including bandgap mode, regulators' drive strength and voltage level) in low power mode at a time.

Note: Enable/disable LVDs/HVDs before invoking this function.

Note: This function will check input parameters based on hardware restrictions before setting registers, if input parameters do not satisfy hardware restrictions the specific error will be reported.

Note: Some hardware restrictions not covered, application should be aware of this and follow this hardware restrictions otherwise some unknown issue may occur:

- a. If Core LDO's drive strength are set to same value in both Active mode and low power mode, the voltage level should also set to same value.
 - b. When switching Core LDO's drive strength from low to normal, ensure the LDO_CORE high voltage level is set to same level that was set prior to switching to the LDO_CORE drive strength. Otherwise, if the LVDs are enabled, an unexpected LVD can occur.
-

Note: If this function can not satisfy some tricky settings, please invoke other APIs in low-level function group.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_lowpower_mode_regulators_config_t` structure.

Return values

- `kStatus_Success` – Config regulators in Low power mode successful.
- `kStatus_SPC_BandgapModeWrong` – The bandgap should not be disabled based on input settings.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `kStatus_SPC_CORELDOLowVoltageWrong` – Core LDO and System LDO do not have same voltage level.

`static inline void SPC_DisableLowPowerModeVddCoreGlitchDetect(SPC_Type *base, bool disable)`
Disable/Enable VDD Core Glitch Detect in low power mode.

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
 - **true** Disable VDD Core Low Voltage detect;
 - **false** Enable VDD Core Low Voltage detect.

`static inline bool SPC_CheckLowPowerModeVddCoreGlitchDetectEnabled(SPC_Type *base)`
Check if Glitch detect hardware is enabled in low power mode.

Parameters

- base – SPC peripheral base address.

Returns

Indicate if Glitch detector is enabled.

```
static inline void SPC_EnableLowPowerModeAnalogModules(SPC_Type *base, uint32_t
                                                    maskValue)
```

Enables analog modules in low power modes.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to enable in low power modes, should be OR'ed value of `spc_analog_module_control`.

```
static inline void SPC_DisableLowPowerModeAnalogModules(SPC_Type *base, uint32_t
                                                    maskValue)
```

Disables analog modules in low power modes.

Parameters

- base – SPC peripheral base address.
- maskValue – The mask of analog modules to disable in low power modes, should be OR'ed value of `spc_analog_module_control`.

```
static inline uint32_t SPC_GetLowPowerModeEnabledAnalogModules(SPC_Type *base)
```

Gets enabled analog modules that enabled in low power modes.

Parameters

- base – SPC peripheral base address.

Returns

The mask of enabled analog modules that enabled in low power modes.

```
static inline uint32_t SPC_GetVoltageDetectStatusFlag(SPC_Type *base)
```

Get Voltage Detect Status Flags.

Parameters

- base – SPC peripheral base address.

Returns

Voltage Detect Status Flags. See `_spc_voltage_detect_flags` for details.

```
static inline void SPC_ClearVoltageDetectStatusFlag(SPC_Type *base, uint8_t mask)
```

Clear Voltage Detect Status Flags.

Parameters

- base – SPC peripheral base address.
- mask – The mask of the voltage detect status flags. See `_spc_voltage_detect_flags` for details.

```
void SPC_SetCoreVoltageDetectConfig(SPC_Type *base, const spc_core_voltage_detect_config_t
                                     *config)
```

Configs CORE voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_core_voltage_detect_config_t` structure.

static inline void SPC_LockCoreVoltageDetectResetSetting(SPC_Type *base)

Locks Core voltage detect reset setting.

This function locks core voltage detect reset setting. After invoking this function any configuration of Core voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockCoreVoltageDetectResetSetting(SPC_Type *base)

Unlocks Core voltage detect reset setting.

This function unlocks core voltage detect reset setting. If locks the Core voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

status_t SPC_EnableActiveModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Active mode.

Note: If the CORE_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core LVD. true - Enable Core Low voltage detector in active mode. false - Disable Core Low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Low Power mode.

This function enables/disables the Core Low Voltage Detector. If enabled the Core Low Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core Low voltage detector in low power mode. false - Disable Core Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect in low power mode successfully.

status_t SPC_EnableActiveModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core High Voltage Detector in Active mode.

Note: If the CORE_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in active mode. false - Disable Core High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable Core High Voltage Detect successfully.

status_t SPC_EnableLowPowerModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core High Voltage Detector in Low Power mode.

This function enables/disables the Core High Voltage Detector. If enabled the Core High Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in low power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in low power mode. false - Disable Core High voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core High Voltage Detect in low power mode successfully.

void SPC_SetSystemVDDLowVoltageLevel(SPC_Type *base, *spc_low_voltage_level_select_t* level)

Set system VDD Low-voltage level selection.

This function selects the system VDD low-voltage level. Changing system VDD low-voltage level must be done after disabling the System VDD low voltage reset and interrupt.

Deprecated:

In latest RM, reserved for all devices, will removed in next release.

Parameters

- base – SPC peripheral base address.
- level – System VDD Low-Voltage level selection.

void SPC_SetSystemVoltageDetectConfig(SPC_Type *base, const *spc_system_voltage_detect_config_t* *config)

Configs SYS voltage detect options.

This function config SYS voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_system_voltage_detect_config_t` structure.

`static inline void SPC_LockSystemVoltageDetectResetSetting(SPC_Type *base)`

Lock System voltage detect reset setting.

This function locks system voltage detect reset setting. After invoking this function any configuration of System Voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

`static inline void SPC_UnlockSystemVoltageDetectResetSetting(SPC_Type *base)`

Unlock System voltage detect reset setting.

This function unlocks system voltage detect reset setting. If locks the System voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

`status_t SPC_EnableActiveModeSystemHighVoltageDetect(SPC_Type *base, bool enable)`

Enables/Disables the System High Voltage Detector in Active mode.

Note: If the System_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in active mode. false - Disable System High voltage detector in active mode.

Return values

`kStatus__Success` – Enable/Disable System High Voltage Detect successfully.

`status_t SPC_EnableActiveModeSystemLowVoltageDetect(SPC_Type *base, bool enable)`

Enables/Disable the System Low Voltage Detector in Active mode.

Note: If the System_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.

- enable – Enable/Disable System LVD. true - Enable System Low voltage detector in active mode. false - Disable System Low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable the System Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeSystemHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System High Voltage Detector in Low Power mode.

Note: If the System_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in low power mode. false - Disable System High voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable System High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeSystemLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System Low Voltage Detector in Low Power mode.

Note: If the System_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System Low voltage detector in low power mode. false - Disable System Low voltage detector in low power mode.

Return values

kStatus_Success – Enables System Low Voltage Detect in low power mode successfully.

void SPC_SetIOVDDLowVoltageLevel(SPC_Type *base, *spc_low_voltage_level_select_t* level)

Set IO VDD Low-Voltage level selection.

This function selects the IO VDD Low-voltage level. Changing IO VDD low-voltage level must be done after disabling the IO VDD low voltage reset and interrupt.

Parameters

- base – SPC peripheral base address.
- level – IO VDD Low-voltage level selection.

void SPC_SetIOVoltageDetectConfig(SPC_Type *base, const *spc_io_voltage_detect_config_t* *config)

Configs IO voltage detect options.

This function config IO voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to `spc_voltage_detect_config_t` structure.

`static inline void SPC_LockIOVoltageDetectResetSetting(SPC_Type *base)`

Lock IO Voltage detect reset setting.

This function locks IO voltage detect reset setting. After invoking this function any configuration of system voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

`static inline void SPC_UnlockIOVoltageDetectResetSetting(SPC_Type *base)`

Unlock IO voltage detect reset setting.

This function unlocks IO voltage detect reset setting. If locks the IO voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

`status_t SPC_EnableActiveModeIOHighVoltageDetect(SPC_Type *base, bool enable)`

Enables/Disables the IO High Voltage Detector in Active mode.

Note: If the IO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in active mode. false - Disable IO High voltage detector in active mode.

Return values

`kStatus_Success` – Enable/Disable IO High Voltage Detect successfully.

`status_t SPC_EnableActiveModeIOLowVoltageDetect(SPC_Type *base, bool enable)`

Enables/Disables the IO Low Voltage Detector in Active mode.

Note: If the IO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in active mode. false - Disable IO Low voltage detector in active mode.

Return values

`kStatus_Success` – Enable IO Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeIOHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO High Voltage Detector in Low Power mode.

Note: If the IO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in low power mode. false - Disable IO High voltage detector in low power mode.

Return values

kStatus_Success – Enable IO High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO Low Voltage Detector in Low Power mode.

Note: If the IO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in low power mode. false - Disable IO Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable IO Low Voltage Detect in low power mode successfully.

void SPC_SetExternalVoltageDomainsConfig(SPC_Type *base, uint8_t lowPowerIsoMask, uint8_t IsoMask)

Configs external voltage domains.

This function configs external voltage domains isolation.

Parameters

- base – SPC peripheral base address.
- lowPowerIsoMask – The mask of external domains isolate enable during low power mode. Please read the Reference Manual for the Bitmap.
- IsoMask – The mask of external domains isolate. Please read the Reference Manual for the Bitmap.

static inline uint8_t SPC_GetExternalDomainsStatus(SPC_Type *base)

Gets External Domains status.

Parameters

- base – SPC peripheral base address.

Returns

The status of each external domain.

```
static inline void SPC_EnableCoreLDORegulator(SPC_Type *base, bool enable)
    Enable/Disable Core LDO regulator.
```

Note: The CORE LDO enable bit is write-once.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CORE LDO Regulator. true - Enable CORE LDO Regulator. false - Disable CORE LDO Regulator.

```
static inline void SPC_PullDownCoreLDORegulator(SPC_Type *base, bool pulldown)
    Enable/Disable the CORE LDO Regulator pull down in Deep Power Down.
```

Note: This function only useful when enabled the CORE LDO Regulator.

Parameters

- base – SPC peripheral base address.
- pulldown – Enable/Disable CORE LDO pulldown in Deep Power Down mode. true - CORE LDO Regulator will discharge in Deep Power Down mode. false - CORE LDO Regulator will not discharge in Deep Power Down mode.

```
status_t SPC_SetActiveModeCoreLDORegulatorConfig(SPC_Type *base, const
                                                    spc_active_mode_core_ldo_option_t
                                                    *option)
```

Configs Core LDO Regulator in Active mode.

Note: The bandgap must be enabled before invoking this function.

Note: To set Core LDO as low drive strength, all HVDs/LVDs must be disabled previously.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_core_ldo_option_t structure.

Return values

- kStatus_Success – Config Core LDO regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_BandgapModeWrong – Bandgap should be enabled before invoking this function.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – To set Core LDO as low drive strength, all LVDs/HVDs must be disabled before invoking this function.

```
status_t SPC_SetActiveModeCoreLDORegulatorVoltageLevel(SPC_Type *base,
                                                         spc_core_ldo_voltage_level_t
                                                         voltageLevel)
```

Set Core LDO Regulator Voltage level in Active mode.

Note: In active mode, the Core LDO voltage level should only be changed when the Core LDO is in normal drive strength.

Note: Update Core LDO voltage level will set Busy flag, this function return only when busy flag is cleared by hardware

Parameters

- base – SPC peripheral base address.
- voltageLevel – Specify the voltage level of CORE LDO Regulator in Active mode, please refer to `spc_core_ldo_voltage_level_t`.

Return values

- `kStatus_SPC_CORELDOVoltageSetFail` – The drive strength of Core LDO is not normal.
- `kStatus_Success` – Set Core LDO regulator voltage level in Active power mode successful.

```
static inline spc_core_ldo_voltage_level_t SPC_GetActiveModeCoreLDOVDDVoltageLevel(SPC_Type
                                                                                   *base)
```

Gets CORE LDO Regulator Voltage level.

This function returns the voltage level of CORE LDO Regulator in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

Voltage level of CORE LDO in type of `spc_core_ldo_voltage_level_t` enumeration.

```
status_t SPC_SetActiveModeCoreLDORegulatorDriveStrength(SPC_Type *base,
                                                         spc_core_ldo_drive_strength_t
                                                         driveStrength)
```

Set Core LDO VDD Regulator Drive Strength in Active mode.

Parameters

- base – SPC peripheral base address.
- driveStrength – Specify the drive strength of CORE LDO Regulator in Active mode, please refer to `spc_core_ldo_drive_strength_t`.

Return values

- `kStatus_Success` – Set Core LDO regulator drive strength in Active power mode successful.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – If any voltage detect enabled, core_ldo's drive strength can not set to low.
- `kStatus_SPC_BandgapModeWrong` – The selected bandgap mode is not allowed.

```
static inline spc_core_ldo_drive_strength_t SPC_GetActiveModeCoreLDODriveStrength(SPC_Type
                                                                                   *base)
```

Gets CORE LDO VDD Regulator Drive Strength in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

Drive Strength of CORE LDO regulator in Active mode, please refer to `spc_core_ldo_drive_strength_t`.

```
status_t SPC_SetLowPowerModeCoreLDORegulatorConfig(SPC_Type *base, const  
                                                    spc_lowpower_mode_core_ldo_option_t  
                                                    *option)
```

Configs CORE LDO Regulator in low power mode.

This function configs CORE LDO Regulator in Low Power mode. If CORE LDO VDD Drive Strength is set to Normal, the CORE LDO VDD regulator voltage level in Active mode must be equal to the voltage level in Low power mode. And the Bandgap must be programmed to select bandgap enabled. Core VDD voltage levels for the Core LDO low power regulator can only be changed when the CORE LDO Drive Strength set as Normal.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the `spc_lowpower_mode_core_ldo_option_t` structure.

Return values

- `kStatus_Success` – Config Core LDO regulator in power mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.
- `#kStatus_SPC_CORELDIVoltageSetFail` – Fail to change Core LDO voltage level.

```
status_t SPC_SetLowPowerModeCoreLDORegulatorVoltageLevel(SPC_Type *base,  
                                                         spc_core_ldo_voltage_level_t  
                                                         voltageLevel)
```

Set Core LDO VDD Regulator Voltage level in Low power mode.

Note: If CORE LDO's drive strength is set to Normal, the CORE LDO VDD regulator voltage in active mode and low power mode must be same.

Note: Voltage level for the CORE LDO in low power mode can only be changed when the CORE LDO Drive Strength set as Normal.

Parameters

- base – SPC peripheral base address.
- voltageLevel – Voltage level of CORE LDO Regulator in Low power mode, please refer to `spc_core_ldo_voltage_level_t`.

Return values

- `kStatus_SPC_CORELDIVoltageWrong` – Voltage level in active mode and low power mode is not same.
- `kStatus_Success` – Set Core LDO regulator voltage level in Low power mode successful.

- `kStatus_SPC_CORELDOVoltageSetFail` – Fail to update voltage level because drive strength is incorrect.

static inline *spc_core_ldo_voltage_level_t* SPC_GetLowPowerCoreLDOVDDVoltageLevel(*SPC_Type* *base)

Gets the CORE LDO VDD Regulator Voltage Level for Low Power modes.

Parameters

- `base` – SPC peripheral base address.

Returns

The CORE LDO VDD Regulator's voltage level.

status_t SPC_SetLowPowerModeCoreLDORegulatorDriveStrength(*SPC_Type* *base,
spc_core_ldo_drive_strength_t driveStrength)

Set Core LDO VDD Regulator Drive Strength in Low power mode.

Parameters

- `base` – SPC peripheral base address.
- `driveStrength` – Specify drive strength of CORE LDO in low power mode.

Return values

- `kStatus_SPC_CORELDOLowDriveStrengthIgnore` – Some voltage detect enabled, CORE LDO's drive strength can not set as low.
- `kStatus_Success` – Set Core LDO regulator drive strength in Low power mode successful.
- `kStatus_SPC_BandgapModeWrong` – Bandgap is disabled when attempt to set CORE LDO work as normal drive strength.

static inline *spc_core_ldo_drive_strength_t* SPC_GetLowPowerCoreLDOVDDDriveStrength(*SPC_Type* *base)

Gets CORE LDO VDD Drive Strength for Low Power modes.

Parameters

- `base` – SPC peripheral base address.

Returns

The CORE LDO's VDD Drive Strength.

static inline void SPC_EnableSystemLDORegulator(*SPC_Type* *base, bool enable)
Enable/Disable System LDO regulator.

Note: The SYSTEM LDO enable bit is write-once.

Parameters

- `base` – SPC peripheral base address.
- `enable` – Enable/Disable System LDO Regulator. true - Enable System LDO Regulator. false - Disable System LDO Regulator.

static inline void SPC_EnableSystemLDOSinkFeature(*SPC_Type* *base, bool sink)
Enable/Disable current sink feature of System LDO Regulator.

Parameters

- `base` – SPC peripheral base address.

- sink – Enable/Disable current sink feature. true - Enable current sink feature of System LDO Regulator. false - Disable current sink feature of System LDO Regulator.

```
status_t SPC_SetActiveModeSystemLDORegulatorConfig(SPC_Type *base, const
                                                    spc_active_mode_sys_ldo_option_t
                                                    *option)
```

Configs System LDO VDD Regulator in Active mode.

Note: If System LDO VDD Drive Strength is set to Normal, the Bandgap mode in Active mode must be programmed to a value that enables the bandgap.

Note: If any voltage detects are kept enabled, configuration to set System LDO VDD drive strength to low will be ignored.

Note: If select System LDO VDD Regulator voltage level to Over Drive Voltage, the Drive Strength of System LDO VDD Regulator must be set to Normal otherwise the regulator Drive Strength will be forced to Normal.

Note: If select System LDO VDD Regulator voltage level to Over Drive Voltage, the High voltage detect must be disabled. Otherwise it will be fail to regulator to Over Drive Voltage.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_sys_ldo_option_t structure.

Return values

- kStatus_Success – Config System LDO regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_BandgapModeWrong – The bandgap is not enabled before invoking this function.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – HVD of System VDD is not disable before setting to Over Drive voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set System LDO VDD regulator's driver strength to Low will be ignored.

```
status_t SPC_SetActiveModeSystemLDORegulatorVoltageLevel(SPC_Type *base,
                                                          spc_sys_ldo_voltage_level_t
                                                          voltageLevel)
```

Set System LDO Regulator voltage level in Active mode.

Note: The system LDO regulator can only operate at the overdrive voltage level for a limited amount of time for the life of chip.

Parameters

- `base` – SPC peripheral base address.
- `voltageLevel` – Specify the voltage level of System LDO Regulator in Active mode.

Return values

- `kStatus_Success` – Set System LDO Regulator voltage level in Active mode successfully.
- `kStatus_SPC_SYSLDOOverDriveVoltageFail` – Must disable system LDO high voltage detector before specifying overdrive voltage.

```
static inline spc_sys_ldo_voltage_level_t SPC_GetActiveModeSystemLDORegulatorVoltageLevel(SPC_Type
                                                                                       *base)
```

Get System LDO Regulator voltage level in Active mode.

Parameters

- `base` – SPC peripheral base address.

Returns

System LDO Regulator voltage level in Active mode, please refer to `spc_sys_ldo_voltage_level_t`.

```
status_t SPC_SetActiveModeSystemLDORegulatorDriveStrength(SPC_Type *base,
                                                           spc_sys_ldo_drive_strength_t
                                                           driveStrength)
```

Set System LDO Regulator Drive Strength in Active mode.

Parameters

- `base` – SPC peripheral base address.
- `driveStrength` – Specify the drive strength of System LDO Regulator in Active mode.

Return values

- `kStatus_Success` – Set System LDO Regulator drive strength in Active mode successfully.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Attempt to specify low drive strength is ignored due to any voltage detect feature is enabled in active mode.
- `kStatus_SPC_BandgapModeWrong` – Bandgap mode in Active mode must be programmed to a value that enables the bandgap if attempt to specify normal drive strength.

```
static inline spc_sys_ldo_drive_strength_t SPC_GetActiveModeSystemLDORegulatorDriveStrength(SPC_Type
                                                                                       *base)
```

Get System LDO Regulator Drive Strength in Active mode.

Parameters

- `base` – SPC peripheral base address.

Returns

System LDO regulator drive strength in Active mode, please refer to `spc_sys_ldo_drive_strength_t`.

```
status_t SPC_SetLowPowerModeSystemLDORegulatorConfig(SPC_Type *base, const
                                                       spc_lowpower_mode_sys_ldo_option_t
                                                       *option)
```

Configs System LDO regulator in low power modes.

This function configures System LDO regulator in low power modes. If System LDO VDD Regulator Drive strength is set to normal, bandgap mode in low power mode must be programmed to a value that enables the Bandgap. If any High voltage detectors or Low Voltage detectors are kept enabled, configuration to set System LDO Regulator drive strength as Low will be ignored.

Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to `spc_lowpower_mode_sys_ldo_option_t` structure.

Return values

- `kStatus_Success` – Config System LDO regulator in Low Power Mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Set driver strength to low will be ignored.

```
status_t SPC_SetLowPowerModeSystemLDORegulatorDriveStrength(SPC_Type *base,  
                                                             spc_sys_ldo_drive_strength_t  
                                                             driveStrength)
```

Set System LDO Regulator drive strength in Low Power Mode.

Parameters

- `base` – SPC peripheral base address.
- `driveStrength` – Specify the drive strength of System LDO Regulator in Low Power Mode.

Return values

- `kStatus_Success` – Set System LDO Regulator drive strength in Low Power Mode successfully.
- `kStatus_SPC_SYSLDOLowDriveStrengthIgnore` – Attempt to specify low drive strength is ignored due to any voltage detect feature is enabled in low power mode.
- `kStatus_SPC_BandgapModeWrong` – Bandgap mode in low power mode must be programmed to a value that enables the bandgap if attempt to specify normal drive strength.

```
static inline spc_sys_ldo_drive_strength_t SPC_GetLowPowerModeSystemLDORegulatorDriveStrength(SPC_Type  
                                                                                             *base)
```

Get System LDO Regulator drive strength in Low Power Mode.

Parameters

- `base` – SPC peripheral base address.

Returns

System LDO regulator drive strength in Low Power Mode, please refer to `spc_sys_ldo_drive_strength_t`.

```
static inline void SPC_EnableDCDCRegulator(SPC_Type *base, bool enable)  
Enable/Disable DCDC Regulator.
```

Note: The DCDC enable bit is write-once, settings only reset after a POR, LVD, or HVD event.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable DCDC Regulator. true - Enable DCDC Regulator.
false - Disable DCDC Regulator.

void SPC_SetDCDCBurstConfig(SPC_Type *base, *spc_dcdc_burst_config_t* *config)

Config DCDC Burst options.

Parameters

- base – SPC peripheral base address.
- config – Pointer to *spc_dcdc_burst_config_t* structure.

static inline void SPC_TriggerDCDCBurstRequest(SPC_Type *base)

Trigger a software burst request to DCDC.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_CheckDCDCBurstAck(SPC_Type *base)

Check if burst acknowledge flag is asserted.

Parameters

- base – SPC peripheral base address.

Return values

- false – DCDC burst not complete.
- true – DCDC burst complete.

static inline void SPC_ClearDCDCBurstAckFlag(SPC_Type *base)

Clear DCDC burst acknowledge flag.

Parameters

- base – SPC peripheral base address.

void SPC_SetDCDCRefreshCount(SPC_Type *base, uint16_t count)

Set the count value of the reference clock to configure the period of DCDC not active.

Note: This function is only useful when DCDC's drive strength is set as pulse refresh.

Note: The pulse duration(time between on and off) is: reference clock period * (count + 2).

Parameters

- base – SPC peripheral base address.
- count – The count value, 16 bit width.

static inline void SPC_EnableDCDCBleedResistor(SPC_Type *base, bool enable)

Enable a bleed resistor to discharge DCDC output when DCDC is disabled.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable bleed resistor.

```
status_t SPC_SetActiveModeDCDCRegulatorConfig(SPC_Type *base, const
                                              spc_active_mode_dcdc_option_t *option)
```

Configs DCDC_CORE Regulator in Active mode.

Note: When changing the DCDC output voltage level, take care to change the CORE LDO voltage level.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_dcdc_option_t structure.

Return values

- kStatus_Success – Config DCDC regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_BandgapModeWrong – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

```
static inline void SPC_SetActiveModeDCDCRegulatorVoltageLevel(SPC_Type *base,
                                                             spc_dcdc_voltage_level_t
                                                             voltageLevel)
```

Set DCDC_CORE Regulator voltage level in Active mode.

Note: When changing the DCDC output voltage level, take care to change the CORE LDO voltage level.

Parameters

- base – SPC peripheral base address.
- voltageLevel – Specify the DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

```
static inline spc_dcdc_voltage_level_t SPC_GetActiveModeDCDCRegulatorVoltageLevel(SPC_Type
                                                                                     *base)
```

Get DCDC_CORE Regulator voltage level in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

```
status_t SPC_SetActiveModeDCDCRegulatorDriveStrength(SPC_Type *base,
                                                      spc_dcdc_drive_strength_t
                                                      driveStrength)
```

Set DCDC_CORE Regulator drive strength in Active mode.

Note: To set DCDC drive strength as Normal, the bandgap must be enabled.

Parameters

- base – SPC peripheral base address.

- `driveStrength` – Specify the DCDC_CORE regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

Return values

- `kStatus_Success` – Set DCDC_CORE Regulator drive strength in Active mode successfully.
- `kStatus_SPC_BandgapModeWrong` – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

```
static inline spc_dcdc_drive_strength_t SPC_GetActiveModeDCDCRegulatorDriveStrength(SPC_Type  
*base)
```

Get DCDC_CORE Regulator drive strength in Active mode.

Parameters

- `base` – SPC peripheral base address.

Returns

DCDC_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

```
status_t SPC_SetLowPowerModeDCDCRegulatorConfig(SPC_Type *base, const  
*spc_lowpower_mode_dcdc_option_t  
*option)
```

Configs DCDC_CORE Regulator in Low power modes.

Note: If DCDC_CORE Drive Strength is set to Normal, the Bandgap mode in Low Power mode must be programmed to a value that enables the Bandgap.

Note: In Deep Power Down mode, DCDC regulator is always turned off.

Parameters

- `base` – SPC peripheral base address.
- `option` – Pointer to the `spc_lowpower_mode_dcdc_option_t` structure.

Return values

- `kStatus_Success` – Config DCDC regulator in low power mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.
- `kStatus_SPC_BandgapModeWrong` – The bandgap mode setting in Low Power mode is wrong.

```
status_t SPC_SetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type *base,  
*spc_dcdc_drive_strength_t  
driveStrength)
```

Set DCDC_CORE Regulator drive strength in Low power mode.

Note: To set drive strength as normal, the bandgap must be enabled.

Parameters

- `base` – SPC peripheral base address.

- `driveStrength` – Specify the DCDC_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

Return values

- `kStatus_Success` – Set DCDC_CORE Regulator drive strength in Low power mode successfully.
- `kStatus_SPC_BandgapModeWrong` – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

```
static inline spc_dcdc_drive_strength_t SPC_GetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type  
                                                                                      *base)
```

Get DCDC_CORE Regulator drive strength in Low power mode.

Parameters

- `base` – SPC peripheral base address.

Returns

DCDC_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

```
static inline void SPC_SetLowPowerModeDCDCRegulatorVoltageLevel(SPC_Type *base,  
                                                                spc_dcdc_voltage_level_t  
                                                                voltageLevel)
```

Set DCDC_CORE Regulator voltage level in Low power mode.

- a. Configure `ACTIVE_CFG[DCDC_VDD_LVL]` to same level programmed in #1.

Note: To change DCDC level in Low-Power mode:

- a. Configure `LP_CFG[DCDC_VDD_LVL]` to desired level;
 - b. Configure `LP_CFG[DCDC_VDD_DS]` to low driver strength;
-

Note: After invoking this function, the voltage level in active mode(wakeup from low power modes) also changed, if it is necessary, please invoke `SPC_SetActiveModeDCDCRegulatorVoltageLevel()` to change to desired voltage level.

Parameters

- `base` – SPC peripheral base address.
- `voltageLevel` – Specify the DCDC_CORE Regulator voltage level, please refer to `spc_dcdc_voltage_level_t`.

```
static inline spc_dcdc_voltage_level_t SPC_GetLowPowerModeDCDCRegulatorVoltageLevel(SPC_Type  
                                                                                      *base)
```

Get DCDC_CORE Regulator voltage level in Low power mode.

Parameters

- `base` – SPC peripheral base address.

Returns

DCDC_CORE Regulator voltage level, please refer to `spc_dcdc_voltage_level_t`.

FSL_SPC_DRIVER_VERSION

SPC driver version 2.10.0.

SPC status enumeration.

Note: Some device(such as MCXA family) do not equip DCDC or System LDO, please refer to the reference manual to check.

Values:

enumerator kStatus_SPC_Busy

The SPC instance is busy executing any type of power mode transition.

enumerator kStatus_SPC_DCDCLowDriveStrengthIgnore

DCDC Low drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_DCDCPulseRefreshModeIgnore

DCDC Pulse Refresh Mode drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_SYSLDOOverDriveVoltageFail

SYS LDO regulate to Over drive voltage failed for SYS LDO HVD must be disabled.

enumerator kStatus_SPC_SYSLDOLowDriveStrengthIgnore

SYS LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_CORELDOLowDriveStrengthIgnore

CORE LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_BandgapModeWrong

Selected Bandgap Mode wrong.

enumerator kStatus_SPC_CORELDOVoltageWrong

Core LDO voltage is wrong.

enumerator kStatus_SPC_CORELDOVoltageSetFail

Core LDO voltage set fail.

enumerator kStatus_SPC_CORELDOVoltageDetectWrong

Settings of CORE_LDO voltage detection is not allowed.

enumerator kStatus_SPC_DCDCCoreLdoVoltageMisMatch

Target voltage level of DCDC not equal to CORE_LDO.

enum _spc_voltage_detect_flags

Voltage Detect Status Flags.

Values:

enumerator kSPC_IOVDDHighVoltageDetectFlag

IO VDD High-Voltage detect flag.

enumerator kSPC_IOVDDLowVoltageDetectFlag

IO VDD Low-Voltage detect flag.

enumerator kSPC_SystemVDDHighVoltageDetectFlag

System VDD High-Voltage detect flag.

enumerator kSPC_SystemVDDLowVoltageDetectFlag

System VDD Low-Voltage detect flag.

enumerator kSPC_CoreVDDHighVoltageDetectFlag

Core VDD High-Voltage detect flag.

enumerator kSPC_CoreVDDLowVoltageDetectFlag

Core VDD Low-Voltage detect flag.

enum __spc_power_domains

SPC power domain isolation status.

Note: Some devices(such as MCXA family) do not contain WAKE Power Domain, please refer to the reference manual to check.

Values:

enumerator kSPC_MAINPowerDomainRetain

Peripherals and IO pads retain in MAIN Power Domain.

enumerator kSPC_WAKEPowerDomainRetain

Peripherals and IO pads retain in WAKE Power Domain.

enum __spc_analog_module_control

The enumeration of all analog module that can be controlled by SPC in active or low-power modes.

Note: Enumerations may not suitable for all devices, please check the specific device's RM for supported analog modules.

Values:

enumerator kSPC_controlVref

Enable/disable VREF in active or low-power modes.

enumerator kSPC_controlUsb3vDet

Enable/disable USB3V_Det in active or low-power modes.

enumerator kSPC_controlDac0

Enable/disable DAC0 in active or low-power modes.

enumerator kSPC_controlDac1

Enable/disable DAC1 in active or low-power modes.

enumerator kSPC_controlDac2

Enable/disable DAC2 in active or low-power modes.

enumerator kSPC_controlOpamp0

Enable/disable OPAMP0 in active or low-power modes.

enumerator kSPC_controlOpamp1

Enable/disable OPAMP1 in active or low-power modes.

enumerator kSPC_controlOpamp2

Enable/disable OPAMP2 in active or low-power modes.

enumerator kSPC_controlOpamp3

Enable/disable OPAMP3 in active or low-power modes.

enumerator kSPC_controlCmp0

Enable/disable CMP0 in active or low-power modes.

enumerator kSPC_controlCmp1

Enable/disable CMP1 in active or low-power modes.

enumerator kSPC_controlCmp2

Enable/disable CMP2 in active or low-power modes.

enumerator kSPC_controlCmp0Dac

Enable/disable CMP0_DAC in active or low-power modes.

enumerator kSPC_controlCmp1Dac

Enable/disable CMP1_DAC in active or low-power modes.

enumerator kSPC_controlCmp2Dac

Enable/disable CMP2_DAC in active or low-power modes.

enumerator kSPC_controlAllModules

Enable/disable all modules in active or low-power modes.

enum _spc_power_domain_id

The enumeration of spc power domain, the connected power domain is chip specific, please refer to chip's RM for details.

Values:

enumerator kSPC_PowerDomain0

Power domain0, the connected power domain is chip specific.

enumerator kSPC_PowerDomain1

Power domain1, the connected power domain is chip specific.

enum _spc_power_domain_low_power_mode

The enumeration of Power domain's low power mode.

Values:

enumerator kSPC_SleepWithSYSClockRunning

Power domain request SLEEP mode with SYS clock running.

enumerator kSPC_DeepSleepWithSysClockOff

Power domain request deep sleep mode with system clock off.

enumerator kSPC_PowerDownWithSysClockOff

Power domain request power down mode with system clock off.

enumerator kSPC_DeepPowerDownWithSysClockOff

Power domain request deep power down mode with system clock off.

enum _spc_lowPower_request_pin_polarity

SPC low power request output pin polarity.

Values:

enumerator kSPC_HighTruePolarity

Control the High Polarity of the Low Power Request Pin.

enumerator kSPC_LowTruePolarity

Control the Low Polarity of the Low Power Request Pin.

enum _spc_lowPower_request_output_override

SPC low power request output override.

Values:

enumerator kSPC_LowPowerRequestNotForced

Not Forced.

enumerator kSPC_LowPowerRequestReserved
Reserved.

enumerator kSPC_LowPowerRequestForcedLow
Forced Low (Ignore LowPower request output polarity setting.)

enumerator kSPC_LowPowerRequestForcedHigh
Forced High (Ignore LowPower request output polarity setting.)

enum _spc_bandgap_mode
SPC Bandgap mode enumeration in Active mode or Low Power mode.
Values:

enumerator kSPC_BandgapDisabled
Bandgap disabled.

enumerator kSPC_BandgapEnabledBufferDisabled
Bandgap enabled with Buffer disabled.

enumerator kSPC_BandgapEnabledBufferEnabled
Bandgap enabled with Buffer enabled.

enumerator kSPC_BandgapReserved
Reserved.

enum _spc_dcdc_voltage_level
DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

Note: kSPC_DCDC_RetentionVoltage not supported for all power modes.

Values:

enumerator kSPC_DCDC_RetentionVoltage
DCDC_CORE Regulator regulate to retention Voltage(Only supported in low power modes)

enumerator kSPC_DCDC_MidVoltage
DCDC_CORE Regulator regulate to Mid Voltage(1.0V).

enumerator kSPC_DCDC_NormalVoltage
DCDC_CORE Regulator regulate to Normal Voltage(1.1V).

enumerator kSPC_DCDC_OverdriveVoltage
DCDC_CORE Regulator regulate to Safe-Mode Voltage(1.2V).

enum _spc_dcdc_drive_strength
DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

Note: Different drive strength differ in these DCDC characteristics: Maximum load current Quiescent current Transient response.

Values:

enumerator kSPC_DCDC_PulseRefreshMode
DCDC_CORE Regulator Drive Strength set to Pulse Refresh Mode, This enum member is only useful for Low Power Mode config, please note that pluse refresh mode is invalid in SLEEP mode.

enumerator kSPC_DCDC_LowDriveStrength

DCDC_CORE regulator Drive Strength set to low.

enumerator kSPC_DCDC_NormalDriveStrength

DCDC_CORE regulator Drive Strength set to Normal.

enum _spc_sys_ldo_voltage_level

SYS LDO regulator voltage level enumeration in Active mode.

Values:

enumerator kSPC_SysLDO_NormalVoltage

SYS LDO VDD Regulator regulate to Normal Voltage(1.8V).

enumerator kSPC_SysLDO_OverDriveVoltage

SYS LDO VDD Regulator regulate to Over Drive Voltage(2.5V).

enum _spc_sys_ldo_drive_strength

SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.

Values:

enumerator kSPC_SysLDO_LowDriveStrength

SYS LDO VDD regulator Drive Strength set to low.

enumerator kSPC_SysLDO_NormalDriveStrength

SYS LDO VDD regulator Drive Strength set to Normal.

enum _spc_core_ldo_voltage_level

Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

Values:

enumerator kSPC_CoreLDO_UnderDriveVoltage

Deprecated:

, to align with description of latest RM, please use
kSPC_CoreLDO_RetentionVoltage as instead.

enumerator kSPC_CoreLDO_RetentionVoltage

Core LDO VDD regulator regulate to retention voltage, please note that only useful in low power modes and not all devices support this options please refer to devices' RM for details.

enumerator kSPC_CoreLDO_MidDriveVoltage

Core LDO VDD regulator regulate to Mid Drive Voltage.

enumerator kSPC_CoreLDO_NormalVoltage

Core LDO VDD regulator regulate to Normal Voltage.

enumerator kSPC_CoreLDO_OverDriveVoltage

Core LDO VDD regulator regulate to overdrive Voltage.

enum _spc_core_ldo_drive_strength

CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

Values:

enumerator kSPC_CoreLDO_LowDriveStrength

Core LDO VDD regulator Drive Strength set to low.

enumerator kSPC_CoreLDO_NormalDriveStrength

Core LDO VDD regulator Drive Strength set to Normal.

enum `_spc_low_voltage_level_select`
IO VDD Low-Voltage Level Select.

Values:

enumerator `kSPC_LowVoltageNormalLevel`

Deprecated:

, please use `kSPC_LowVoltageHighRange` as instead.

enumerator `kSPC_LowVoltageSafeLevel`

Deprecated:

, please use `kSPC_LowVoltageLowRange` as instead.

enumerator `kSPC_LowVoltageHighRange`

High range LVD threshold.

enumerator `kSPC_LowVoltageLowRange`

Low range LVD threshold.

enum `_spc_vdd_core_glitch_ripple_counter_select`

Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

Values:

enumerator `kSPC_selectBit0Of4bitRippleCounter`

Select bit-0 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator `kSPC_selectBit1Of4bitRippleCounter`

Select bit-1 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator `kSPC_selectBit2Of4bitRippleCounter`

Select bit-2 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator `kSPC_selectBit3Of4bitRippleCounter`

Select bit-3 of 4-bit Ripple Counter to detect glitch on VDD Core.

enum `_spc_sram_operate_voltage`

The list of the operating voltage for the SRAM's read/write timing margin.

Values:

enumerator `kSPC_sramOperateAt1P0V`

SRAM configured for 1.0V operation.

enumerator `kSPC_sramOperateAt1P1V`

SRAM configured for 1.1V operation.

enumerator `kSPC_sramOperateAt1P2V`

SRAM configured for 1.2V operation.

typedef enum `_spc_power_domain_id` `spc_power_domain_id_t`

The enumeration of spc power domain, the connected power domain is chip specific, please refer to chip's RM for details.

typedef enum `_spc_power_domain_low_power_mode` `spc_power_domain_low_power_mode_t`

The enumeration of Power domain's low power mode.

typedef enum `_spc_lowPower_request_pin_polarity` `spc_lowpower_request_pin_polarity_t`

SPC low power request output pin polarity.

typedef enum `_spc_lowPower_request_output_override` `spc_lowpower_request_output_override_t`

SPC low power request output override.

typedef enum *_spc_bandgap_mode* spc_bandgap_mode_t

SPC Bandgap mode enumeration in Active mode or Low Power mode.

typedef enum *_spc_dcdc_voltage_level* spc_dcdc_voltage_level_t

DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

Note: kSPC_DCDC_RetentionVoltage not supported for all power modes.

typedef enum *_spc_dcdc_drive_strength* spc_dcdc_drive_strength_t

DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

Note: Different drive strength differ in these DCDC characteristics: Maximum load current
Quiescent current Transient response.

typedef enum *_spc_sys_ldo_voltage_level* spc_sys_ldo_voltage_level_t

SYS LDO regulator voltage level enumeration in Active mode.

typedef enum *_spc_sys_ldo_drive_strength* spc_sys_ldo_drive_strength_t

SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.

typedef enum *_spc_core_ldo_voltage_level* spc_core_ldo_voltage_level_t

Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

typedef enum *_spc_core_ldo_drive_strength* spc_core_ldo_drive_strength_t

CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

typedef enum *_spc_low_voltage_level_select* spc_low_voltage_level_select_t

IO VDD Low-Voltage Level Select.

typedef enum *_spc_vdd_core_glitch_ripple_counter_select*

spc_vdd_core_glitch_ripple_counter_select_t

Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

typedef enum *_spc_sram_operate_voltage* spc_sram_operate_voltage_t

The list of the operating voltage for the SRAM's read/write timing margin.

typedef struct *_spc_vdd_core_glitch_detector_config* spc_vdd_core_glitch_detector_config_t

The configuration of VDD Core glitch detector.

typedef struct *_spc_sram_voltage_config* spc_sram_voltage_config_t

typedef struct *_spc_lowpower_request_config* spc_lowpower_request_config_t

Low Power Request output pin configuration.

typedef struct *_spc_active_mode_core_ldo_option* spc_active_mode_core_ldo_option_t

Core LDO regulator options in Active mode.

typedef struct *_spc_active_mode_sys_ldo_option* spc_active_mode_sys_ldo_option_t

System LDO regulator options in Active mode.

typedef struct *_spc_active_mode_dcdc_option* spc_active_mode_dcdc_option_t

DCDC regulator options in Active mode.

typedef struct *_spc_lowpower_mode_core_ldo_option* spc_lowpower_mode_core_ldo_option_t

Core LDO regulator options in Low Power mode.

typedef struct *_spc_lowpower_mode_sys_ldo_option* spc_lowpower_mode_sys_ldo_option_t

System LDO regulator options in Low Power mode.

typedef struct *_spc_lowpower_mode_dcdc_option* spc_lowpower_mode_dcdc_option_t
DCDC regulator options in Low Power mode.

typedef struct *_spc_dcdc_burst_config* spc_dcdc_burst_config_t
DCDC Burst configuration.

Deprecated:

Do not recommend to use this structure.

typedef struct *_spc_voltage_detect_option* spc_voltage_detect_option_t
CORE/SYS/IO VDD Voltage Detect options.

typedef struct *_spc_core_voltage_detect_config* spc_core_voltage_detect_config_t
Core Voltage Detect configuration.

typedef struct *_spc_system_voltage_detect_config* spc_system_voltage_detect_config_t
System Voltage Detect Configuration.

typedef struct *_spc_io_voltage_detect_config* spc_io_voltage_detect_config_t
IO Voltage Detect Configuration.

typedef struct *_spc_active_mode_regulators_config* spc_active_mode_regulators_config_t
Active mode configuration.

typedef struct *_spc_lowpower_mode_regulators_config* spc_lowpower_mode_regulators_config_t
Low Power Mode configuration.

SPC_EVD_CFG_REG_EVDISO_SHIFT

SPC_EVD_CFG_REG_EVDLPISO_SHIFT

SPC_EVD_CFG_REG_EVDSTAT_SHIFT

SPC_EVD_CFG_REG_EVDISO(x)

SPC_EVD_CFG_REG_EVDLPISO(x)

SPC_EVD_CFG_REG_EVDSTAT(x)

struct *_spc_vdd_core_glitch_detector_config*
#include <fsl_spc.h> The configuration of VDD Core glitch detector.

Public Members

spc_vdd_core_glitch_ripple_counter_select_t rippleCounterSelect
Used to set ripple counter.

uint8_t resetTimeoutValue
The timeout value used to reset glitch detect/compare logic after an initial glitch is detected.

bool enableReset
Used to enable/disable POR/LVD reset that caused by CORE VDD glitch detect error.

bool enableInterrupt
Used to enable/disable hardware interrupt if CORE VDD glitch detect error.

struct *_spc_sram_voltage_config*
#include <fsl_spc.h>

Public Members

spc_sram_operate_voltage_t operateVoltage

Specifies the operating voltage for the SRAM's read/write timing margin.

bool requestVoltageUpdate

Used to control whether request an SRAM trim value change.

struct _spc_lowpower_request_config

#include <fsl_spc.h> Low Power Request output pin configuration.

Public Members

bool enable

Low Power Request Output enable.

spc_lowpower_request_pin_polarity_t polarity

Low Power Request Output pin polarity select.

spc_lowpower_request_output_override_t override

Low Power Request Output Override.

struct _spc_active_mode_core_ldo_option

#include <fsl_spc.h> Core LDO regulator options in Active mode.

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage

Core LDO Regulator Voltage Level selection in Active mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength

Core LDO Regulator Drive Strength selection in Active mode

struct _spc_active_mode_sys_ldo_option

#include <fsl_spc.h> System LDO regulator options in Active mode.

Public Members

spc_sys_ldo_voltage_level_t SysLDOVoltage

System LDO Regulator Voltage Level selection in Active mode.

spc_sys_ldo_drive_strength_t SysLDODriveStrength

System LDO Regulator Drive Strength selection in Active mode.

struct _spc_active_mode_dcdc_option

#include <fsl_spc.h> DCDC regulator options in Active mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage

DCDC Regulator Voltage Level selection in Active mode.

spc_dcdc_drive_strength_t DCDCDriveStrength

DCDC_CORE Regulator Drive Strength selection in Active mode.

struct _spc_lowpower_mode_core_ldo_option

#include <fsl_spc.h> Core LDO regulator options in Low Power mode.

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage

Core LDO Regulator Voltage Level selection in Low Power mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength

Core LDO Regulator Drive Strength selection in Low Power mode

struct *_spc_lowpower_mode_sys_ldo_option*

#include <fsl_spc.h> System LDO regulator options in Low Power mode.

Public Members

spc_sys_ldo_drive_strength_t SysLDODriveStrength

System LDO Regulator Drive Strength selection in Low Power mode.

struct *_spc_lowpower_mode_dcdc_option*

#include <fsl_spc.h> DCDC regulator options in Low Power mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage

DCDC Regulator Voltage Level selection in Low Power mode.

spc_dcdc_drive_strength_t DCDCDriveStrength

DCDC_CORE Regulator Drive Strength selection in Low Power mode.

struct *_spc_dcdc_burst_config*

#include <fsl_spc.h> DCDC Burst configuration.

Deprecated:

Do not recommend to use this structure.

Public Members

bool softwareBurstRequest

Enable/Disable DCDC Software Burst Request.

bool externalBurstRequest

Enable/Disable DCDC External Burst Request.

bool stabilizeBurstFreq

Enable/Disable DCDC frequency stabilization.

uint8_t freq

The frequency of the current burst.

struct *_spc_voltage_detect_option*

#include <fsl_spc.h> CORE/SYS/IO VDD Voltage Detect options.

Public Members

bool HVDInterruptEnable

CORE/SYS/IO VDD High Voltage Detect interrupt enable.

bool HVDRResetEnable

CORE/SYS/IO VDD High Voltage Detect reset enable.

bool LVDInterruptEnable

CORE/SYS/IO VDD Low Voltage Detect interrupt enable.

bool LVDRResetEnable

CORE/SYS/IO VDD Low Voltage Detect reset enable.

struct __spc_core_voltage_detect_config

#include <fsl_spc.h> Core Voltage Detect configuration.

Public Members

spc_voltage_detect_option_t option

Core VDD Voltage Detect option.

struct __spc_system_voltage_detect_config

#include <fsl_spc.h> System Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option

System VDD Voltage Detect option.

spc_low_voltage_level_select_t level

Deprecated:

, reserved for all devices, will removed in next release.

struct __spc_io_voltage_detect_config

#include <fsl_spc.h> IO Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option

IO VDD Voltage Detect option.

spc_low_voltage_level_select_t level

IO VDD Low-voltage level selection.

struct __spc_active_mode_regulators_config

#include <fsl_spc.h> Active mode configuration.

Public Members

spc_bandgap_mode_t bandgapMode

Specify bandgap mode in active mode.

bool lpBuff

Enable/disable CMP bandgap buffer.

spc_active_mode_dcdc_option_t DCDCOption

Specify DCDC configurations in active mode.

spc_active_mode_sys_ldo_option_t SysLDOOption

Specify System LDO configurations in active mode.

spc_active_mode_core_ldo_option_t CoreLDOOption
Specify Core LDO configurations in active mode.

struct *_spc_lowpower_mode_regulators_config*
#include <fsl_spc.h> Low Power Mode configuration.

Public Members

bool *lpIREF*
Enable/disable low power IREF in low power modes.

spc_bandgap_mode_t bandgapMode
Specify bandgap mode in low power modes.

bool *lpBuff*
Enable/disable CMP bandgap buffer in low power modes.

bool *CoreIVS*
Enable/disable CORE VDD internal voltage scaling.

spc_lowpower_mode_dcdc_option_t DCDCOption
Specify DCDC configurations in low power modes.

spc_lowpower_mode_sys_ldo_option_t SysLDOOption
Specify system LDO configurations in low power modes.

spc_lowpower_mode_core_ldo_option_t CoreLDOOption
Specify core LDO configurations in low power modes.

2.62 MCX_VBAT: Smart Power Switch

The enumeration of VBAT module status.

Values:

enumerator *kStatus_VBAT_Fro16kNotEnabled*
Internal 16kHz free running oscillator not enabled.

enumerator *kStatus_VBAT_BandgapNotEnabled*
Bandgap not enabled.

enumerator *kStatus_VBAT_WrongCapacitanceValue*
Wrong capacitance for selected oscillator mode.

enumerator *kStatus_VBAT_ClockMonitorLocked*
Clock monitor locked.

enumerator *kStatus_VBAT_OSC32KNotReady*
OSC32K not ready.

enumerator *kStatus_VBAT_LDONotReady*
LDO not ready.

enumerator *kStatus_VBAT_TamperLocked*
Tamper locked.

enum _vbat_status_flag

The enumeration of VBAT status flags.

Values:

enumerator kVBAT__StatusFlagPORDetect

VBAT domain has been reset

enumerator kVBAT__StatusFlagWakeupPin

A falling edge is detected on the wakeup pin.

enumerator kVBAT__StatusFlagBandgapTimer0

Bandgap Timer0 period reached.

enumerator kVBAT__StatusFlagBandgapTimer1

Bandgap Timer1 period reached.

enumerator kVBAT__StatusFlagLdoReady

LDO is enabled and ready.

enumerator kVBAT__StatusFlagOsc32kReady

OSC32k is enabled and clock is ready.

enumerator kVBAT__StatusFlagInterrupt0Detect

Interrupt 0 asserted.

enumerator kVBAT__StatusFlagInterrupt1Detect

Interrupt 1 asserted.

enumerator kVBAT__StatusFlagInterrupt2Detect

Interrupt 2 asserted.

enumerator kVBAT__StatusFlagInterrupt3Detect

Interrupt 2 asserted.

enum _vbat_interrupt_enable

The enumeration of VBAT interrupt enable.

Values:

enumerator kVBAT__InterruptEnablePORDetect

Enable POR detect interrupt.

enumerator kVBAT__InterruptEnableWakeupPin

Enable the interrupt when a falling edge is detected on the wakeup pin.

enumerator kVBAT__InterruptEnableBandgapTimer0

Enable the interrupt if Bandgap Timer0 period reached.

enumerator kVBAT__InterruptEnableBandgapTimer1

Enable the interrupt if Bandgap Timer1 period reached.

enumerator kVBAT__InterruptEnableLdoReady

Enable LDO ready interrupt.

enumerator kVBAT__InterruptEnableOsc32kReady

Enable OSC32K ready interrupt.

enumerator kVBAT__InterruptEnableInterrupt0

Enable the interrupt0.

enumerator kVBAT__InterruptEnableInterrupt1

Enable the interrupt1.

enumerator kVBAT__InterruptEnableInterrupt2

Enable the interrupt2.

enumerator kVBAT__InterruptEnableInterrupt3

Enable the interrupt3.

enumerator kVBAT__AllInterruptsEnable

Enable all interrupts.

enum __vbat__wakeup__enable

The enumeration of VBAT wakeup enable.

Values:

enumerator kVBAT__WakeupEnablePORDetect

Enable POR detect wakeup.

enumerator kVBAT__WakeupEnableWakeupPin

Enable wakeup feature when a falling edge is detected on the wakeup pin.

enumerator kVBAT__WakeupEnableBandgapTimer0

Enable wakeup feature when bandgap timer0 period reached.

enumerator kVBAT__WakeupEnableBandgapTimer1

Enable wakeup feature when bandgap timer1 period reached.

enumerator kVBAT__WakeupEnableLdoReady

Enable wakeup when LDO ready.

enumerator kVBAT__WakeupEnableOsc32kReady

Enable wakeup when OSC32k ready.

enumerator kVBAT__WakeupEnableInterrupt0

Enable wakeup when interrupt0 asserted.

enumerator kVBAT__WakeupEnableInterrupt1

Enable wakeup when interrupt1 asserted.

enumerator kVBAT__WakeupEnableInterrupt2

Enable wakeup when interrupt2 asserted.

enumerator kVBAT__WakeupEnableInterrupt3

Enable wakeup when interrupt3 asserted.

enumerator kVBAT__AllWakeupsEnable

Enable all wakeup.

enum __vbat__tamper__enable

The enumeration of VBAT tamper enable.

Values:

enumerator kVBAT__TamperEnablePOR

Enable tamper if POR asserted in STATUS register.

enumerator kVBAT__TamperEnableClockDetect

Enable tamper if clock monitor detect an error.

enumerator kVBAT__TamperEnableConfigDetect

Enable tamper if configuration error detected.

enumerator kVBAT__TamperEnableVoltageDetect

Enable tamper if voltage monitor detect an error.

enumerator kVBAT_TamperEnableTemperatureDetect
Enable tamper if temperature monitor detect an error.

enumerator kVBAT_TamperEnableSec0Detect
Enable tamper if security input 0 detect an error.

enum __vbat_bandgap_timer_id
The enumeration of bandgap timer id, VBAT support two bandgap timers.
Values:

enumerator kVBAT_BandgapTimer0
Bandgap Timer0.

enumerator kVBAT_BandgapTimer1
Bandgap Timer1.

enum __vbat_clock_enable
The enumeration of connections for OSC32K/FRO32K output clock to other modules.
Values:

enumerator kVBAT_EnableClockToDomain0
Enable clock to power domain0.

enumerator kVBAT_EnableClockToDomain1
Enable clock to power domain1.

enumerator kVBAT_EnableClockToDomain2
Enable clock to power domain2.

enumerator kVBAT_EnableClockToDomain3
Enable clock to power domain3.

enum __vbat_ram_array
The enumeration of SRAM arrays that controlled by VBAT. .
Values:

enumerator kVBAT_SramArray0
Specify SRAM array0 that controlled by VBAT.

enumerator kVBAT_SramArray1
Specify SRAM array1 that controlled by VBAT.

enumerator kVBAT_SramArray2
Specify SRAM array2 that controlled by VBAT.

enumerator kVBAT_SramArray3
Specify SRAM array3 that controlled by VBAT.

enum __vbat_bandgap_refresh_period
The enumeration of bandgap refresh period.
Values:

enumerator kVBAT_BandgapRefresh7P8125ms
Bandgap refresh every 7.8125ms.

enumerator kVBAT_BandgapRefresh15P625ms
Bandgap refresh every 15.625ms.

enumerator kVBAT_BandgapRefresh31P25ms
Bandgap refresh every 31.25ms.

enumerator kVBAT_BandgapRefresh62P5ms

Bandgap refresh every 62.5ms.

enum _vbat_bandgap_timer0_timeout_period

The enumeration of bandgap timer0 timeout period.

Values:

enumerator kVBAT_BangapTimer0Timeout1s

Bandgap timer0 timerout every 1s.

enumerator kVBAT_BangapTimer0Timeout500ms

Bandgap timer0 timerout every 500ms.

enumerator kVBAT_BangapTimer0Timeout250ms

Bandgap timer0 timerout every 250ms.

enumerator kVBAT_BangapTimer0Timeout125ms

Bandgap timer0 timerout every 125ms.

enumerator kVBAT_BangapTimer0Timeout62P5ms

Bandgap timer0 timerout every 62.5ms.

enumerator kVBAT_BangapTimer0Timeout31P25ms

Bandgap timer0 timerout every 31.25ms.

enum _vbat_osc32k_operate_mode

The enumeration of osc32k operate mode, including Bypass mode, low power switched mode and so on.

Values:

enumerator kVBAT_Osc32kEnabledToTransconductanceMode

Set to transconductance mode.

enumerator kVBAT_Osc32kEnabledToLowPowerBackupMode

Set to low power backup mode.

enumerator kVBAT_Osc32kEnabledToLowPowerSwitchedMode

Set to low power switched mode.

enum _vbat_osc32k_load_capacitance_select

The enumeration of OSC32K load capacitance.

Values:

enumerator kVBAT_Osc32kCrystalLoadCap0pF

Internal capacitance bank is enabled, set the internal capacitance to 0 pF.

enumerator kVBAT_Osc32kCrystalLoadCap2pF

Internal capacitance bank is enabled, set the internal capacitance to 2 pF.

enumerator kVBAT_Osc32kCrystalLoadCap4pF

Internal capacitance bank is enabled, set the internal capacitance to 4 pF.

enumerator kVBAT_Osc32kCrystalLoadCap6pF

Internal capacitance bank is enabled, set the internal capacitance to 6 pF.

enumerator kVBAT_Osc32kCrystalLoadCap8pF

Internal capacitance bank is enabled, set the internal capacitance to 8 pF.

enumerator kVBAT_Osc32kCrystalLoadCap10pF

Internal capacitance bank is enabled, set the internal capacitance to 10 pF.

enumerator kVBAT_Osc32kCrystalLoadCap12pF

Internal capacitance bank is enabled, set the internal capacitance to 12 pF.

enumerator kVBAT_Osc32kCrystalLoadCap14pF

Internal capacitance bank is enabled, set the internal capacitance to 14 pF.

enumerator kVBAT_Osc32kCrystalLoadCap16pF

Internal capacitance bank is enabled, set the internal capacitance to 16 pF.

enumerator kVBAT_Osc32kCrystalLoadCap18pF

Internal capacitance bank is enabled, set the internal capacitance to 18 pF.

enumerator kVBAT_Osc32kCrystalLoadCap20pF

Internal capacitance bank is enabled, set the internal capacitance to 20 pF.

enumerator kVBAT_Osc32kCrystalLoadCap22pF

Internal capacitance bank is enabled, set the internal capacitance to 22 pF.

enumerator kVBAT_Osc32kCrystalLoadCap24pF

Internal capacitance bank is enabled, set the internal capacitance to 24 pF.

enumerator kVBAT_Osc32kCrystalLoadCap26pF

Internal capacitance bank is enabled, set the internal capacitance to 26 pF.

enumerator kVBAT_Osc32kCrystalLoadCap28pF

Internal capacitance bank is enabled, set the internal capacitance to 28 pF.

enumerator kVBAT_Osc32kCrystalLoadCap30pF

Internal capacitance bank is enabled, set the internal capacitance to 30 pF.

enumerator kVBAT_Osc32kCrystalLoadCapBankDisabled

Internal capacitance bank is disabled.

enum _vbat_osc32k_start_up_time

The enumeration of start-up time of the oscillator.

Values:

enumerator kVBAT_Osc32kStartUpTime8Sec

Configure the start-up time as 8 seconds.

enumerator kVBAT_Osc32kStartUpTime4Sec

Configure the start-up time as 4 seconds.

enumerator kVBAT_Osc32kStartUpTime2Sec

Configure the start-up time as 2 seconds.

enumerator kVBAT_Osc32kStartUpTime1Sec

Configure the start-up time as 1 seconds.

enumerator kVBAT_Osc32kStartUpTime0P5Sec

Configure the start-up time as 0.5 seconds.

enumerator kVBAT_Osc32kStartUpTime0P25Sec

Configure the start-up time as 0.25 seconds.

enumerator kVBAT_Osc32kStartUpTime0P125Sec

Configure the start-up time as 0.125 seconds.

enumerator kVBAT_Osc32kStartUpTime0P5MSec

Configure the start-up time as 0.5 milliseconds.

enum `_vbat_internal_module_supply`

The enumeration of VBAT module supplies.

Values:

enumerator `kVBAT_ModuleSuppliedByVddBat`

VDD_BAT supplies VBAT modules.

enumerator `kVBAT_ModuleSuppliedByVddSys`

VDD_SYS supplies VBAT modules.

enum `_vbat_clock_monitor_divide_trim`

The enumeration of VBAT clock monitor divide trim value.

Values:

enumerator `kVBAT_ClockMonitorOperateAt1kHz`

Clock monitor operates at 1 kHz.

enumerator `kVBAT_ClockMonitorOperateAt64Hz`

Clock monitor operates at 64 Hz.

enum `_vbat_clock_monitor_freq_trim`

The enumeration of VBAT clock monitor frequency trim value used to adjust the clock monitor assert.

Values:

enumerator `kVBAT_ClockMonitorAssert2Cycle`

Clock monitor assert 2 cycles after expected edge.

enumerator `kVBAT_ClockMonitorAssert4Cycle`

Clock monitor assert 4 cycles after expected edge.

enumerator `kVBAT_ClockMonitorAssert6Cycle`

Clock monitor assert 8 cycles after expected edge.

enumerator `kVBAT_ClockMonitorAssert8Cycle`

Clock monitor assert 8 cycles after expected edge.

typedef enum `_vbat_bandgap_refresh_period` `vbat_bandgap_refresh_period_t`

The enumeration of bandgap refresh period.

typedef enum `_vbat_bandgap_timer0_timeout_period` `vbat_bandgap_timer0_timeout_period_t`

The enumeration of bandgap timer0 timeout period.

typedef enum `_vbat_osc32k_operate_mode` `vbat_osc32k_operate_mode_t`

The enumeration of osc32k operate mode, including Bypass mode, low power switched mode and so on.

typedef enum `_vbat_osc32k_load_capacitance_select` `vbat_osc32k_load_capacitance_select_t`

The enumeration of OSC32K load capacitance.

typedef enum `_vbat_osc32k_start_up_time` `vbat_osc32k_start_up_time_t`

The enumeration of start-up time of the oscillator.

typedef enum `_vbat_internal_module_supply` `vbat_internal_module_supply_t`

The enumeration of VBAT module supplies.

typedef enum `_vbat_clock_monitor_divide_trim` `vbat_clock_monitor_divide_trim_t`

The enumeration of VBAT clock monitor divide trim value.

```
typedef enum _vbat_clock_monitor_freq_trim vbat_clock_monitor_freq_trim_t
```

The enumeration of VBAT clock monitor frequency trim value used to adjust the clock monitor assert.

```
typedef struct _vbat_fro16k_config vbat_fro16k_config_t
```

The structure of internal 16kHz free running oscillator attributes.

```
typedef struct _vbat_clock_monitor_config vbat_clock_monitor_config_t
```

The structure of internal clock monitor, including divide trim and frequency trim.

```
typedef struct _vbat_tamper_config vbat_tamper_config_t
```

The structure of Tamper configuration.

```
FSL_VBAT_DRIVER_VERSION
```

VBAT driver version 2.5.0.

```
VBAT_LDORAMC_RET_MASK
```

```
VBAT_LDORAMC_RET_SHIFT
```

```
VBAT_LDORAMC_RET(x)
```

```
kVBAT_EnableClockToVddBat
```

```
kVBAT_EnableClockToVddSys
```

```
kVBAT_EnableClockToVddWake
```

```
kVBAT_EnableClockToVddMain
```

```
void VBAT_ConfigFRO16k(VBAT_Type *base, const vbat_fro16k_config_t *config)
```

Configure internal 16kHz free running oscillator, including enable FRO16k, gate FRO16k output.

Parameters

- base – VBAT peripheral base address.
- config – Pointer to vbat_fro16k_config_t structure.

```
static inline void VBAT_EnableFRO16k(VBAT_Type *base, bool enable)
```

Enable/disable internal 16kHz free running oscillator.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable 16kHz FRO.
 - **true** Enable internal 16kHz free running oscillator.
 - **false** Disable internal 16kHz free running oscillator.

```
static inline bool VBAT_CheckFRO16kEnabled(VBAT_Type *base)
```

Check if internal 16kHz free running oscillator is enabled.

Parameters

- base – VBAT peripheral base address.

Return values

- true – The internal 16kHz Free running oscillator is enabled.
- false – The internal 16kHz Free running oscillator is disabled.

```
static inline void VBAT__UngateFRO16k(VBAT_Type *base, uint8_t connectionsMask)
```

Enable FRO16kHz output clock to selected modules.

Parameters

- base – VBAT peripheral base address.
- connectionsMask – The mask of modules that FRO16k is connected, should be the OR'ed value of vbat_clock_enable_t.

```
static inline void VBAT__GateFRO16k(VBAT_Type *base, uint8_t connectionsMask)
```

Disable FRO16kHz output clock to selected modules.

Parameters

- base – VBAT peripheral base address.
- connectionsMask – The OR'ed value of vbat_clock_enable_t.

```
static inline void VBAT__LockFRO16kSettings(VBAT_Type *base)
```

Lock settings of internal 16kHz free running oscillator, please note that if locked 16kHz FRO's settings can not be updated until the next POR.

Note: Please note that the operation to ungate/gate FRO 16kHz output clock can not be locked by this function.

Parameters

- base – VBAT peripheral base address.

```
static inline bool VBAT__CheckFRO16kSettingsLocked(VBAT_Type *base)
```

Check if FRO16K settings are locked.

Parameters

- base – VBAT peripheral base address.

Returns

true in case of FRO16k settings are locked, false in case of FRO16k settings are not locked.

```
static inline void VBAT__EnableCrystalOsc32k(VBAT_Type *base, bool enable)
```

Enable/disable 32K Crystal Oscillator.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable 32k Crystal Oscillator:
 - **true** Enable crystal oscillator and polling status register to check clock is ready.
 - **false** Disable crystal oscillator.

```
static inline void VBAT__BypassCrystalOsc32k(VBAT_Type *base, bool enableBypass)
```

Bypass 32k crystal oscillator, the clock is still output by oscillator but this clock is the same as clock provided on EXTAL pin.

Note: In bypass mode, oscillator must be enabled; To exit bypass mode, oscillator must be disabled.

Parameters

- base – VBAT peripheral base address.
- enableBypass – Used to enter/exit bypass mode:
 - **true** Enter into bypass mode;
 - **false** Exit bypass mode.

```
static inline void VBAT__AdjustCrystalOsc32kAmplifierGain(VBAT_Type *base, uint8_t coarse,  
                                                         uint8_t fine)
```

Adjust 32k crystal oscillator amplifier gain.

Parameters

- base – VBAT peripheral base address.
- coarse – Specify amplifier coarse trim value.
- fine – Specify amplifier fine trim value.

```
status_t VBAT__SetCrystalOsc32kModeAndLoadCapacitance(VBAT_Type *base,  
                                                       vbat_osc32k_operate_mode_t  
                                                       operateMode,  
                                                       vbat_osc32k_load_capacitance_select_t  
                                                       xtalCap,  
                                                       vbat_osc32k_load_capacitance_select_t  
                                                       extalCap)
```

Set 32k crystal oscillator mode and load capacitance for the XTAL/EXTAL pin.

Parameters

- base – VBAT peripheral base address.
- operateMode – Specify the crystal oscillator mode, please refer to `vbat_osc32k_operate_mode_t`.
- xtalCap – Specify the internal capacitance for the XTAL pin from the capacitor bank.
- extalCap – Specify the internal capacitance for the EXTAL pin from the capacitor bank.

Return values

- `kStatus_VBAT__WrongCapacitanceValue` – The load capacitance value to set is not align with operate mode's requirements.
- `kStatus_Success` – Success to set operate mode and load capacitance.

```
static inline void VBAT__TrimCrystalOsc32kStartupTime(VBAT_Type *base,  
                                                       vbat_osc32k_start_up_time_t  
                                                       startupTime)
```

Trim 32k crystal oscillator startup time.

Parameters

- base – VBAT peripheral base address.
- startupTime – Specify the startup time of the oscillator.

```
static inline void VBAT__SetOsc32kSwitchModeComparatorTrimValue(VBAT_Type *base, uint8_t  
                                                                comparatorTrimValue)
```

Set crystal oscillator comparator trim value when oscillator is set as low power switch mode.

Parameters

- base – VBAT peripheral base address.

- `comparatorTrimValue` – Comparator trim value, ranges from 0 to 7.

```
static inline void VBAT_SetOsc32kSwitchModeDelayTrimValue(VBAT_Type *base, uint8_t
                                                         delayTrimValue)
```

Set crystal oscillator delay trim value when oscillator is set as low power switch mode.

Parameters

- `base` – VBAT peripheral base address.
- `delayTrimValue` – Delay trim value, ranges from 0 to 15.

```
static inline void VBAT_SetOsc32kSwitchModeCapacitorTrimValue(VBAT_Type *base, uint8_t
                                                             capacitorTrimValue)
```

Set crystal oscillator capacitor trim value when oscillator is set as low power switch mode.

Parameters

- `base` – VBAT peripheral base address.
- `capacitorTrimValue` – Capacitor value to trim, ranges from 0 to 3.

```
static inline void VBAT_LockOsc32kSettings(VBAT_Type *base)
```

Lock Osc32k settings, after locked all writes to the Oscillator registers are blocked.

Parameters

- `base` – VBAT peripheral base address.

```
static inline void VBAT_UnlockOsc32kSettings(VBAT_Type *base)
```

Unlock Osc32k settings.

Parameters

- `base` – VBAT peripheral base address.

```
static inline bool VBAT_CheckOsc32kSettingsLocked(VBAT_Type *base)
```

Check if osc32k settings are locked.

Parameters

- `base` – VBAT peripheral base address.

Returns

true in case of osc32k settings are locked, false in case of osc32k settings are not locked.

```
static inline void VBAT_UngateOsc32k(VBAT_Type *base, uint8_t connectionsMask)
```

Enable OSC32k output clock to selected modules.

Parameters

- `base` – VBAT peripheral base address.
- `connectionsMask` – The OR'ed value of `vbat_clock_enable_t`.

```
static inline void VBAT_GateOsc32k(VBAT_Type *base, uint8_t connectionsMask)
```

Disable OSC32k output clock to selected modules.

Parameters

- `base` – VBAT peripheral base address.
- `connectionsMask` – The OR'ed value of `vbat_clock_enable_t`.

status_t VBAT_EnableBandgap(VBAT_Type *base, bool enable)
Enable/disable Bandgap.

Note: The FRO16K must be enabled before enabling the bandgap.

Note: This setting can be locked by VBAT_LockRamLdoSettings() function.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap.
 - **true** Enable the bandgap.
 - **false** Disable the bandgap.

Return values

- kStatus_Success – Success to enable/disable the bandgap.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable the bandgap due to FRO16k is not enabled previously.

static inline bool VBAT_CheckBandgapEnabled(VBAT_Type *base)
Check if bandgap is enabled.

Parameters

- base – VBAT peripheral base address.

Return values

- true – The bandgap is enabled.
- false – The bandgap is disabled.

static inline void VBAT_EnableBandgapRefreshMode(VBAT_Type *base, bool
enableRefreshMode)

Enable/disable bandgap low power refresh mode.

Note: For lowest power consumption, refresh mode must be enabled.

Note: This setting can be locked by VBAT_LockRamLdoSettings() function.

Parameters

- base – VBAT peripheral base address.
- enableRefreshMode – Used to enable/disable bandgap low power refresh mode.
 - **true** Enable bandgap low power refresh mode.
 - **false** Disable bandgap low power refresh mode.

status_t VBAT_EnableBackupSRAMRegulator(VBAT_Type *base, bool enable)
Enable/disable Backup RAM Regulator(RAM_LDO).

Note: This setting can be locked by VBAT_LockRamLdoSettings() function.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable RAM_LDO.
 - **true** Enable backup SRAM regulator.
 - **false** Disable backup SRAM regulator.

Return values

- kStatusSuccess – Success to enable/disable backup SRAM regulator.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable backup SRAM regulator due to FRO16k is not enabled previously.
- kStatus_VBAT_BandgapNotEnabled – Fail to enable backup SRAM regulator due to the bandgap is not enabled previously.

static inline void VBAT_LockRamLdoSettings(VBAT_Type *base)

Lock settings of RAM_LDO, please note that if locked then RAM_LDO's settings can not be updated until the next POR.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT_CheckRamLdoSettingsLocked(VBAT_Type *base)

Check if RAM_LDO settings is locked.

Parameters

- base – VBAT peripheral base address.

Returns

true in case of RAM_LDO settings are locked, false in case of RAM_LDO settings are unlocked.

status_t VBAT_SwitchSRAMPowerByLDOSRAM(VBAT_Type *base)

Switch the SRAM to be powered by LDO_RAM.

Note: This function can be used to switch the SRAM to the VBAT retention supply at any time, but please note that the SRAM must not be accessed during this time.

Note: Invoke this function to switch power supply before switching off external power.

Note: RAM_LDO must be enabled before invoking this function.

Note: To access the SRAM arrays retained by the LDO_RAM, please invoke VBAT_SwitchSRAMPowerBySocSupply(), after external power is switched back on.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatusSuccess – Success to Switch SRAM powered by VBAT.
- kStatus_VBAT_Fro16kNotEnabled – Fail to switch SRAM powered by VBAT due to FRO16K not enabled previously.

```
static inline void VBAT_SwitchSRAMPowerBySocSupply(VBAT_Type *base)
```

Switch the RAM to be powered by Soc Supply in software mode.

Parameters

- base – VBAT peripheral base address.

```
static inline void VBAT_PowerOffSRAMsInLowPowerModes(VBAT_Type *base, uint8_t  
sramMask)
```

Power off selected SRAM array in low power modes.

Parameters

- base – VBAT peripheral base address.
- sramMask – The mask of SRAM array to power off, should be the OR'ed value of vbat_ram_array_t.

```
static inline void VBAT_RetainSRAMsInLowPowerModes(VBAT_Type *base, uint8_t sramMask)
```

Retain selected SRAM array in low power modes.

Parameters

- base – VBAT peripheral base address.
- sramMask – The mask of SRAM array to retain, should be the OR'ed value of vbat_ram_array_t.

```
static inline void VBAT_EnableSRAMIsolation(VBAT_Type *base, bool enable)
```

Enable/disable SRAM isolation.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable SRAM violation.
 - **true** SRAM will be isolated.
 - **false** SRAM state follows the SoC power modes.

```
status_t VBAT_EnableBandgapTimer(VBAT_Type *base, bool enable, uint8_t timerIdMask)
```

Enable/disable Bandgap timer.

Note: The bandgap timer is available when the bandgap is enabled and are clocked by the FRO16k.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap timer.
- timerIdMask – The mask of bandgap timer Id, should be the OR'ed value of vbat_bandgap_timer_id_t.

Return values

- kStatus_Success – Success to enable/disable selected bandgap timer.

- `kStatus_VBAT_Fro16kNotEnabled` – Fail to enable/disable selected bandgap timer due to FRO16k not enabled previously.
- `kStatus_VBAT_BandgapNotEnabled` – Fail to enable/disable selected bandgap timer due to bandgap not enabled previously.

```
void VBAT_SetBandgapTimer0TimeoutValue(VBAT_Type *base,  
                                       vbat_bandgap_timer0_timeout_period_t  
                                       timeoutPeriod)
```

Set bandgap timer0 timeout value.

Note: The timeout value can only be changed when the timer is disabled.

Parameters

- `base` – VBAT peripheral base address.
- `timeoutPeriod` – Bandgap timer timeout value, please refer to `vbat_bandgap_timer0_timeout_period_t`.

```
void VBAT_SetBandgapTimer1TimeoutValue(VBAT_Type *base, uint32_t timeoutPeriod)  
Set bandgap timer1 timeout value.
```

Note: The timeout value can only be changed when the timer is disabled.

Parameters

- `base` – VBAT peripheral base address.
- `timeoutPeriod` – The bandgap timerout 1 period, in number of seconds, ranging from 0 to 65535s.

```
static inline void VBAT_SwitchVBATModuleSupplyActiveMode(VBAT_Type *base,  
                                                         vbat_internal_module_supply_t  
                                                         supply)
```

Control the VBAT internal switch in active mode, VBAT modules can be supplied by VDD_BAT and VDD_SYS.

Parameters

- `base` – VBAT peripheral base address.
- `supply` – Used to control the VBAT internal switch.

```
static inline vbat_internal_module_supply_t VBAT_GetVBATModuleSupply(VBAT_Type *base)  
Get VBAT module supply in active mode.
```

Parameters

- `base` – VBAT peripheral base address.

Returns

VDD_SYS supplies VBAT modules or VDD_BAT supplies VBAT modules, in type of `vbat_internal_module_supply_t`.

```
static inline void VBAT_SwitchVBATModuleSupplyLowPowerMode(VBAT_Type *base,  
                                                           vbat_internal_module_supply_t  
                                                           supply)
```

Control the VBAT internal switch in low power modes.

Note: If VBAT modules are supplied by VDD_SYS in low power modes, VBAT module will also supplied by VDD_SYS in active mode.

Parameters

- base – VBAT peripheral base address.
- supply – Used to specify which voltage input supply VBAT modules in low power mode.

static inline void VBAT_LockSwitchControl(VBAT_Type *base)

Lock switch control, if locked all writes to the switch registers will be blocked.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_UnlockSwitchControl(VBAT_Type *base)

Unlock switch control.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT_CheckSwitchControlLocked(VBAT_Type *base)

Check if switch control is locked.

Parameters

- base – VBAT peripheral base address.

Return values

- false – switch control is not locked.
- true – switch control is locked, any writes to related registers are blocked.

status_t VBAT_InitClockMonitor(VBAT_Type *base, const vbat_clock_monitor_config_t *config)

Initialize the VBAT clock monitor, enable clock monitor and set the clock monitor configuration.

Note: Both FRO16K and OSC32K should be enabled and stable before invoking this function.

Parameters

- base – VBAT peripheral base address.
- config – Pointer to vbat_clock_monitor_config_t structure.

Return values

- kStatus_Success – Clock monitor is initialized successfully.
- kStatus_VBAT_Fro16kNotEnabled – FRO16K is not enabled.
- kStatus_VBAT_Osc32kNotReady – OSC32K is not ready.
- kStatus_VBAT_ClockMonitorLocked – Clock monitor is locked.

status_t VBAT_DeinitMonitor(VBAT_Type *base)

Deinitialize the VBAT clock monitor.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatus_Success – Clock monitor is de-initialized successfully.
- kStatus_VBAT_ClockMonitorLocked – Control of Clock monitor is locked.

```
static inline void VBAT__EnableClockMonitor(VBAT_Type *base, bool enable)
```

Enable/disable clock monitor.

- false: disable clock monitor.

Parameters

- base – VBAT peripheral base address.
- enable – Switcher to enable/disable clock monitor:
 - true: enable clock monitor;

```
static inline void VBAT__SetClockMonitorDivideTrim(VBAT_Type *base,  
                                                    vbat_clock_monitor_divide_trim_t  
                                                    divideTrim)
```

Set clock monitor's divide trim, available value is kVBAT_ClockMonitorOperateAt1kHz and kVBAT_ClockMonitorOperateAt64Hz.

Parameters

- base – VBAT peripheral base address.
- divideTrim – Specify divide trim value, please refer to vbat_clock_monitor_divide_trim_t.

```
static inline void VBAT__SetClockMonitorFrequencyTrim(VBAT_Type *base,  
                                                       vbat_clock_monitor_freq_trim_t  
                                                       freqTrim)
```

Set clock monitor's frequency trim, available value is kVBAT_ClockMonitorAssert2Cycle, kVBAT_ClockMonitorAssert4Cycle, kVBAT_ClockMonitorAssert6Cycle and kVBAT_ClockMonitorAssert8Cycle.

Parameters

- base – VBAT peripheral base address.
- freqTrim – Specify frequency trim value, please refer to vbat_clock_monitor_freq_trim_t.

```
static inline void VBAT__LockClockMonitorControl(VBAT_Type *base)
```

Lock clock monitor enable/disable control.

Note: If locked, it is not allowed to change clock monitor enable/disable control.

Parameters

- base – VBAT peripheral base address.

```
static inline void VBAT__UnlockClockMonitorControl(VBAT_Type *base)
```

Unlock clock monitor enable/disable control.

Parameters

- base – VBAT peripheral base address.

static inline bool VBAT__CheckClockMonitorControlLocked(VBAT_Type *base)

Check if clock monitor enable/disable control is locked.

Note: If locked, it is not allowed to change clock monitor enable/disable control.

Parameters

- base – VBAT peripheral base address.

Return values

- false – clock monitor enable/disable control is not locked.
- true – clock monitor enable/disable control is locked, any writes to related registers are blocked.

status_t VBAT__InitTamper(VBAT_Type *base, const vbat_tamper_config_t *config)

Initialize tamper control.

Note: Both FRO16K and bandgap should be enabled before calling this function.

Parameters

- base – VBAT peripheral base address.
- config – Pointer to vbat_tamper_config_t structure.

Return values

- kStatus_Success – Tamper is initialized successfully.
- kStatus_VBAT__TamperLocked – Tamper control is locked.
- kStatus_VBAT__BandgapNotEnabled – Bandgap is not enabled.
- kStatus_VBAT__Fro16kNotEnabled – FRO 16K is not enabled.

status_t VBAT__DeinitTamper(VBAT_Type *base)

De-initialize tamper control.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatus_Success – Tamper is de-initialized successfully.
- kStatus_VBAT__TamperLocked – Tamper control is locked.

static inline void VBAT__EnableTamper(VBAT_Type *base, uint32_t tamperEnableMask)

Enable tampers for VBAT.

Parameters

- base – VBAT peripheral base address.
- tamperEnableMask – Mask of tamper to be enabled, should be the OR'ed value of _vbat_tamper_enable.

static inline void VBAT__DisableTamper(VBAT_Type *base, uint32_t tamperEnableMask)

Disable tampers for VBAT.

Parameters

- base – VBAT peripheral base address.

- `tamperEnableMask` – Mask of tamper to be disabled, should be the OR'ed value of `_vbat_tamper_enable`.

static inline uint32_t VBAT_GetTamperEnableInfo(VBAT_Type *base)

Get tamper enable information.

Parameters

- `base` – VBAT peripheral base address.

Returns

Mask of tamper enable information, should be the OR'ed value of `_vbat_tamper_enable`.

static inline void VBAT_LockTamperControl(VBAT_Type *base)

Lock tamper control, if locked, it is not allowed to change tamper control.

Parameters

- `base` – VBAT peripheral base address.

static inline void VBAT_UnlockTamperControl(VBAT_Type *base)

Unlock tamper control.

Parameters

- `base` – VBAT peripheral base address.

static inline bool VBAT_CheckTamperControlLocked(VBAT_Type *base)

Check if tamper control is locked.

Parameters

- `base` – VBAT peripheral base address.

Return values

- `false` – Tamper control is not locked.
- `true` – Tamper control is locked, any writes to related registers are blocked.

static inline uint32_t VBAT_GetStatusFlags(VBAT_Type *base)

Get VBAT status flags.

Parameters

- `base` – VBAT peripheral base address.

Returns

The asserted status flags, should be the OR'ed value of `vbat_status_flag_t`.

static inline void VBAT_ClearStatusFlags(VBAT_Type *base, uint32_t mask)

Clear VBAT status flags.

Parameters

- `base` – VBAT peripheral base address.
- `mask` – The mask of status flags to be cleared, should be the OR'ed value of `vbat_status_flag_t` except `kVBAT_StatusFlagLdoReady`, `kVBAT_StatusFlagOsc32kReady`, `kVBAT_StatusFlagInterrupt0Detect`, `kVBAT_StatusFlagInterrupt1Detect`, `kVBAT_StatusFlagInterrupt2Detect`, `kVBAT_StatusFlagInterrupt3Detect`.

static inline void VBAT_EnableInterrupts(VBAT_Type *base, uint32_t mask)

Enable interrupts for the VBAT module, such as POR detect interrupt, Wakeup Pin interrupt and so on.

Parameters

- `base` – VBAT peripheral base address.
- `mask` – The mask of interrupts to be enabled, should be the OR'ed value of `vbat_interrupt_enable_t`.

static inline void VBAT_DisableInterrupts(VBAT_Type *base, uint32_t mask)

Disable interrupts for the VBAT module, such as POR detect interrupt, wakeup pin interrupt and so on.

Parameters

- `base` – VBAT peripheral base address.
- `mask` – The mask of interrupts to be disabled, should be the OR'ed value of `vbat_interrupt_enable_t`.

static inline void VBAT_EnableWakeup(VBAT_Type *base, uint32_t mask)

Enable wakeup for the VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- `base` – VBAT peripheral base address.
- `mask` – The mask of enumerators in `vbat_wakeup_enable_t`.

static inline void VBAT_DisableWakeup(VBAT_Type *base, uint32_t mask)

Disable wakeup for VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- `base` – VBAT peripheral base address.
- `mask` – The mask of enumerators in `vbat_wakeup_enable_t`.

static inline void VBAT_LockInterruptWakeupSettings(VBAT_Type *base)

Lock VBAT interrupt and wakeup settings, please note that if locked the interrupt and wakeup settings can not be updated until the next POR.

Parameters

- `base` – VBAT peripheral base address.

static inline void VBAT_SetWakeupPinDefaultState(VBAT_Type *base, bool assert)

Set the default state of the WAKEUP_b pin output when no enabled wakeup source is asserted.

Parameters

- `base` – VBAT peripheral base address.
- `assert` – Used to set default state of the WAKEUP_b pin output:
 - **true** WAKEUP_b output state is logic one;
 - **false** WAKEUP_b output state is logic zero.

struct __vbat_fro16k_config

#include <fsl_vbat.h> The structure of internal 16kHz free running oscillator attributes.

Public Members

bool enableFRO16k

Enable/disable internal 16kHz free running oscillator.

uint8_t enabledConnectionsMask

The mask of connected modules to enable FRO16k clock output.

struct __vbat_clock_monitor_config

#include <fsl_vbat.h> The structure of internal clock monitor, including divide trim and frequency trim.

Public Members

vbat_clock_monitor_freq_trim_t freqTrim

Frequency trim value used to adjust the clock monitor assert, please refer to vbat_clock_monitor_freq_trim_t.

bool lock

Lock the clock monitor control after enabled.

struct __vbat_tamper_config

#include <fsl_vbat.h> The structure of Tamper configuration.

Public Members

bool enableVoltageDetect

Enable/disable voltage detection.

bool enableTemperatureDetect

Enable/disable temperature detection.

bool lock

Lock the tamper control after enabled.

2.63 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

2.64 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.

2.65 OTP: One-Time Programmable memory and API

FSL_OTP_DRIVER_VERSION

OTP driver version 2.0.1.

Current version: 2.0.1

Change log:

- Version 2.0.1
 - Fixed MISRA-C 2012 violations.
- Version 2.0.0
 - Initial version.

enum *_otp_bank*

Bank bit flags.

Values:

enumerator kOTP_Bank0

Bank 0.

enumerator kOTP_Bank1

Bank 1.

enumerator kOTP_Bank2

Bank 2.

enumerator kOTP_Bank3

Bank 3.

enum __otp_word

Bank word bit flags.

Values:

enumerator kOTP_Word0

Word 0.

enumerator kOTP_Word1

Word 1.

enumerator kOTP_Word2

Word 2.

enumerator kOTP_Word3

Word 3.

enum __otp_lock

Lock modifications of a read or write access to a bank register.

Values:

enumerator kOTP_LockDontLock

Do not lock.

enumerator kOTP_LockLock

Lock till reset.

enum __otp_status

OTP error codes.

Values:

enumerator kStatus_OTP_WrEnableInvalid

Write enable invalid.

enumerator kStatus_OTP_SomeBitsAlreadyProgrammed

Some bits already programmed.

enumerator kStatus_OTP_AllDataOrMaskZero

All data or mask zero.

enumerator kStatus_OTP_WriteAccessLocked

Write access locked.

enumerator kStatus_OTP_ReadDataMismatch

Read data mismatch.

enumerator kStatus_OTP_UsbIdEnabled

USB ID enabled.

enumerator kStatus_OTP_EthMacEnabled

Ethernet MAC enabled.

enumerator kStatus_OTP_AesKeysEnabled

AES keys enabled.

enumerator kStatus_OTP_IllegalBank

Illegal bank.

enumerator `kStatus_OTP_ShufflerConfigNotValid`

Shuffler config not valid.

enumerator `kStatus_OTP_ShufflerNotEnabled`

Shuffler not enabled.

enumerator `kStatus_OTP_ShufflerCanOnlyProgSingleKey`

Shuffler can only program single key.

enumerator `kStatus_OTP_IllegalProgramData`

Illegal program data.

enumerator `kStatus_OTP_ReadAccessLocked`

Read access locked.

typedef enum *otp_bank* otp_bank_t

Bank bit flags.

typedef enum *otp_word* otp_word_t

Bank word bit flags.

typedef enum *otp_lock* otp_lock_t

Lock modifications of a read or write access to a bank register.

static inline *status_t* OTP_Init(void)

Initializes OTP controller.

Returns

`kStatus_Success` upon successful execution, error status otherwise.

static inline *status_t* OTP_EnableBankWriteMask(*otp_bank_t* bankMask)

Unlock one or more OTP banks for write access.

Parameters

- bankMask – bit flag that specifies which banks to unlock.

Returns

`kStatus_Success` upon successful execution, error status otherwise.

static inline *status_t* OTP_DisableBankWriteMask(*otp_bank_t* bankMask)

Lock one or more OTP banks for write access.

Parameters

- bankMask – bit flag that specifies which banks to lock.

Returns

`kStatus_Success` upon successful execution, error status otherwise.

static inline *status_t* OTP_EnableBankWriteLock(uint32_t bankIndex, *otp_word_t* regEnableMask, *otp_word_t* regDisableMask, *otp_lock_t* lockWrite)

Locks or unlocks write access to a register of an OTP bank and possibly lock un/locking of it.

Parameters

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regEnableMask – bit flag that specifies for which words to enable writing.
- regDisableMask – bit flag that specifies for which words to disable writing.
- lockWrite – specifies if access set can be modified or is locked till reset.

Returns

kStatus_Success upon successful execution, error status otherwise.

```
static inline status_t OTP__EnableBankReadLock(uint32_t bankIndex, otp_word_t
                                             regEnableMask, otp_word_t regDisableMask,
                                             otp_lock_t lockWrite)
```

Locks or unlocks read access to a register of an OTP bank and possibly lock un/locking of it.

Parameters

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regEnableMask – bit flag that specifies for which words to enable reading.
- regDisableMask – bit flag that specifies for which words to disable reading.
- lockWrite – specifies if access set can be modified or is locked till reset.

Returns

kStatus_Success upon successful execution, error status otherwise.

```
static inline status_t OTP__ProgramRegister(uint32_t bankIndex, uint32_t regIndex, uint32_t
                                           value)
```

Program a single register in an OTP bank.

Parameters

- bankIndex – OTP bank index, 0 = bank 0, 1 = bank 1 etc.
- regIndex – OTP register index.
- value – value to write.

Returns

kStatus_Success upon successful execution, error status otherwise.

```
static inline uint32_t OTP__GetDriverVersion(void)
```

Returns the version of the OTP driver in ROM.

Returns

version.

```
FSL_COMPONENT_ID
```

```
__OTP_ERR_BASE
```

```
__OTP_MAKE_STATUS(errorCode)
```

2.66 PDM: Microphone Interface

2.67 PDM Driver

```
void PDM_Init(PDM_Type *base, const pdm_config_t *config)
```

Initializes the PDM peripheral.

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by PDM_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

Parameters

- base – PDM base pointer
- config – PDM configuration structure.

void PDM_Deinit(PDM_Type *base)

De-initializes the PDM peripheral.

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

Parameters

- base – PDM base pointer

static inline void PDM_Reset(PDM_Type *base)

Resets the PDM module.

Parameters

- base – PDM base pointer

static inline void PDM_Enable(PDM_Type *base, bool enable)

Enables/disables PDM interface.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled, false means PDM interface is disabled.

static inline void PDM_EnableDebugMode(PDM_Type *base, bool enable)

Enables/disables debug mode for PDM. The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface enter debug mode, false means PDM interface in normal mode.

static inline void PDM_EnableInDebugMode(PDM_Type *base, bool enable)

Enables/disables PDM interface in debug mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode.

static inline void PDM_EnterLowLeakageMode(PDM_Type *base, bool enable)

Enables/disables PDM interface disable/Low Leakage mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode.

static inline void PDM_EnableChannel(PDM_Type *base, uint8_t channel, bool enable)

Enables/disables the PDM channel.

Parameters

- base – PDM base pointer
- channel – PDM channel number need to enable or disable.
- enable – True means enable PDM channel, false means disable.

```
void PDM_SetChannelConfig(PDM_Type *base, uint32_t channel, const pdm_channel_config_t
                        *config)
```

PDM one channel configurations.

Parameters

- base – PDM base pointer
- config – PDM channel configurations.
- channel – channel number. after completing the current frame in debug mode.

```
status_t PDM_SetSampleRateConfig(PDM_Type *base, uint32_t sourceClock_HZ, uint32_t
                                sampleRate_HZ)
```

PDM set sample rate.

Note: This function is depend on the configuration of the PDM and PDM channel, so the correct call sequence is

```
PDM_Init(base, pdmConfig)
PDM_SetChannelConfig(base, channel, &channelConfig)
PDM_SetSampleRateConfig(base, source, sampleRate)
```

Parameters

- base – PDM base pointer
- sourceClock_HZ – PDM source clock frequency.
- sampleRate_HZ – PDM sample rate.

```
status_t PDM_SetSampleRate(PDM_Type *base, uint32_t enableChannelMask,
                            pdm_df_quality_mode_t qualityMode, uint8_t osr, uint32_t clkDiv)
```

PDM set sample rate.

Deprecated:

Do not use this function. It has been superceded by PDM_SetSampleRateConfig

Parameters

- base – PDM base pointer
- enableChannelMask – PDM channel enable mask.
- qualityMode – quality mode.
- osr – cic oversample rate
- clkDiv – clock divider

```
uint32_t PDM_GetInstance(PDM_Type *base)
```

Get the instance number for PDM.

Parameters

- base – PDM base pointer.

static inline uint32_t PDM_GetStatus(PDM_Type *base)

Gets the PDM internal status flag. Use the Status Mask in `_pdm_internal_status` to get the status value needed.

Parameters

- base – PDM base pointer

Returns

PDM status flag value.

static inline uint32_t PDM_GetFifoStatus(PDM_Type *base)

Gets the PDM FIFO status flag. Use the Status Mask in `_pdm_fifo_status` to get the status value needed.

Parameters

- base – PDM base pointer

Returns

FIFO status.

static inline uint32_t PDM_GetRangeStatus(PDM_Type *base)

Gets the PDM Range status flag. Use the Status Mask in `_pdm_range_status` to get the status value needed.

Parameters

- base – PDM base pointer

Returns

output status.

static inline void PDM_ClearStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status between `kPDM_StatusFrequencyLow` and `kPDM_StatusCh7FifoDataAvaliable`.

static inline void PDM_ClearFIFoStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in `_pdm_fifo_status`.

static inline void PDM_ClearRangeStatus(PDM_Type *base, uint32_t mask)

Clears the PDM range status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in `_pdm_range_status`.

void PDM_EnableInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM interrupt requests.

Parameters

- base – PDM base pointer

- **mask** – interrupt source The parameter can be a combination of the following sources if defined.
 - `kPDM_ErrorInterruptEnable`
 - `kPDM_FIFOInterruptEnable`

static inline void PDM_DisableInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM interrupt requests.

Parameters

- **base** – PDM base pointer
- **mask** – interrupt source The parameter can be a combination of the following sources if defined.
 - `kPDM_ErrorInterruptEnable`
 - `kPDM_FIFOInterruptEnable`

static inline void PDM_EnableDMA(PDM_Type *base, bool enable)

Enables/disables the PDM DMA requests.

Parameters

- **base** – PDM base pointer
- **enable** – True means enable DMA, false means disable DMA.

static inline uint32_t PDM_GetDataRegisterAddress(PDM_Type *base, uint32_t channel)

Gets the PDM data register address.

This API is used to provide a transfer address for the PDM DMA transfer configuration.

Parameters

- **base** – PDM base pointer.
- **channel** – Which data channel used.

Returns

data register address.

void PDM_ReadFifo(PDM_Type *base, uint32_t startChannel, uint32_t channelNums, void *buffer, size_t size, uint32_t dataWidth)

PDM read fifo.

Note: : This function support 16 bit only for IP version that only supports 16bit.

Parameters

- **base** – PDM base pointer.
- **startChannel** – start channel number.
- **channelNums** – total enabled channelnums.
- **buffer** – received buffer address.
- **size** – number of samples to read.
- **dataWidth** – sample width.

void PDM_SetChannelGain(PDM_Type *base, uint32_t channel, *pdm_df_output_gain_t* gain)

Set the PDM channel gain.

Please note for different quality mode, the valid gain value is different, reference RM for detail.

Parameters

- base – PDM base pointer.
- channel – PDM channel index.
- gain – channel gain, the register gain value range is 0 - 15.

void PDM_TransferCreateHandle(PDM_Type *base, *pdm_handle_t* *handle,
pdm_transfer_callback_t callback, void *userData)

Initializes the PDM handle.

This function initializes the handle for the PDM transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

status_t PDM_TransferSetChannelConfig(PDM_Type *base, *pdm_handle_t* *handle, uint32_t
channel, const *pdm_channel_config_t* *config, uint32_t
format)

PDM set channel transfer config.

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- channel – PDM channel.
- config – channel config.
- format – data format, support data width configurations, *pdm_data_width*.

Return values

kStatus_PDM_ChannelConfig_Failed – or kStatus_Success.

status_t PDM_TransferReceiveNonBlocking(PDM_Type *base, *pdm_handle_t* *handle,
pdm_transfer_t *xfer)

Performs an interrupt non-blocking receive transfer on PDM.

Note: This API returns immediately after the transfer initiates. Call the PDM_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_PDM_Busy, the transfer is finished.

Parameters

- base – PDM base pointer
- handle – Pointer to the *pdm_handle_t* structure which stores the transfer state.
- xfer – Pointer to the *pdm_transfer_t* structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_PDM_Busy – Previous receive still not finished.

`void PDM_TransferAbortReceive(PDM_Type *base, pdm_handle_t *handle)`
 Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – PDM base pointer
- `handle` – Pointer to the `pdm_handle_t` structure which stores the transfer state.

`void PDM_TransferHandleIRQ(PDM_Type *base, pdm_handle_t *handle)`
 Tx interrupt handler.

Parameters

- `base` – PDM base pointer.
- `handle` – Pointer to the `pdm_handle_t` structure.

FSL_PDM_DRIVER_VERSION
 Version 2.9.3

PDM return status.

Values:

enumerator `kStatus_PDM_Busy`
 PDM is busy.

enumerator `kStatus_PDM_CLK_LOW`
 PDM clock frequency low

enumerator `kStatus_PDM_FIFO_ERROR`
 PDM FIFO underrun or overflow

enumerator `kStatus_PDM_QueueFull`
 PDM FIFO underrun or overflow

enumerator `kStatus_PDM_Idle`
 PDM is idle

enumerator `kStatus_PDM_Output_ERROR`
 PDM is output error

enumerator `kStatus_PDM_ChannelConfig_Failed`
 PDM channel config failed

enum `_pdm_interrupt_enable`
 The PDM interrupt enable flag.

Values:

enumerator `kPDM_ErrorInterruptEnable`
 PDM channel error interrupt enable.

enumerator `kPDM_FIFOInterruptEnable`
 PDM channel FIFO interrupt

enum _pdm_internal_status

The PDM status.

Values:

enumerator kPDM_StatusDfBusyFlag

Decimation filter is busy processing data

enumerator kPDM_StatusFrequencyLow

Mic app clock frequency not high enough

enumerator kPDM_StatusCh0FifoDataAvaliable

channel 0 fifo data reached watermark level

enumerator kPDM_StatusCh1FifoDataAvaliable

channel 1 fifo data reached watermark level

enumerator kPDM_StatusCh2FifoDataAvaliable

channel 2 fifo data reached watermark level

enumerator kPDM_StatusCh3FifoDataAvaliable

channel 3 fifo data reached watermark level

enum _pdm_channel_enable_mask

PDM channel enable mask.

Values:

enumerator kPDM_EnableChannel0

channgel 0 enable mask

enumerator kPDM_EnableChannel1

channgel 1 enable mask

enumerator kPDM_EnableChannel2

channgel 2 enable mask

enumerator kPDM_EnableChannel3

channgel 3 enable mask

enumerator kPDM_EnableChannelAll

enum _pdm_fifo_status

The PDM fifo status.

Values:

enumerator kPDM_FifoStatusUnderflowCh0

channel0 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh1

channel1 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh2

channel2 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh3

channel3 fifo status underflow

enumerator kPDM_FifoStatusOverflowCh0

channel0 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh1

channel1 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh2
channel2 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh3
channel3 fifo status overflow

enum __pdm_range_status

The PDM output status.

Values:

enumerator kPDM_RangeStatusUnderFlowCh0
channel0 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh1
channel1 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh2
channel2 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh3
channel3 range status underflow

enumerator kPDM_RangeStatusOverFlowCh0
channel0 range status overflow

enumerator kPDM_RangeStatusOverFlowCh1
channel1 range status overflow

enumerator kPDM_RangeStatusOverFlowCh2
channel2 range status overflow

enumerator kPDM_RangeStatusOverFlowCh3
channel3 range status overflow

enum __pdm_dc_removal

PDM DC removal configurations.

Values:

enumerator kPDM_DcRemovalCutOff20Hz
DC removal cut off 20HZ

enumerator kPDM_DcRemovalCutOff13Hz
DC removal cut off 13.3HZ

enumerator kPDM_DcRemovalCutOff40Hz
DC removal cut off 40HZ

enumerator kPDM_DcRemovalBypass
DC removal bypass

enum __pdm_df_quality_mode

PDM decimation filter quality mode.

Values:

enumerator kPDM_QualityModeMedium
quality mode medium

enumerator kPDM_QualityModeHigh
quality mode high

enumerator kPDM_QualityModeLow
quality mode low

enumerator kPDM_QualityModeVeryLow0
quality mode very low0

enumerator kPDM_QualityModeVeryLow1
quality mode very low1

enumerator kPDM_QualityModeVeryLow2
quality mode very low2

enum __pdm_qulaity_mode_k_factor
PDM quality mode K factor.

Values:

enumerator kPDM_QualityModeHighKFactor
high quality mode K factor = 1 / 2

enumerator kPDM_QualityModeMediumKFactor
medium/very low0 quality mode K factor = 2 / 2

enumerator kPDM_QualityModeLowKFactor
low/very low1 quality mode K factor = 4 / 2

enumerator kPDM_QualityModeVeryLow2KFactor
very low2 quality mode K factor = 8 / 2

enum __pdm_df_output_gain
PDM decimation filter output gain.

Values:

enumerator kPDM_DfOutputGain0
Decimation filter output gain 0

enumerator kPDM_DfOutputGain1
Decimation filter output gain 1

enumerator kPDM_DfOutputGain2
Decimation filter output gain 2

enumerator kPDM_DfOutputGain3
Decimation filter output gain 3

enumerator kPDM_DfOutputGain4
Decimation filter output gain 4

enumerator kPDM_DfOutputGain5
Decimation filter output gain 5

enumerator kPDM_DfOutputGain6
Decimation filter output gain 6

enumerator kPDM_DfOutputGain7
Decimation filter output gain 7

enumerator kPDM_DfOutputGain8
Decimation filter output gain 8

enumerator kPDM_DfOutputGain9
Decimation filter output gain 9

```

enumerator kPDM_DfOutputGain10
    Decimation filter output gain 10
enumerator kPDM_DfOutputGain11
    Decimation filter output gain 11
enumerator kPDM_DfOutputGain12
    Decimation filter output gain 12
enumerator kPDM_DfOutputGain13
    Decimation filter output gain 13
enumerator kPDM_DfOutputGain14
    Decimation filter output gain 14
enumerator kPDM_DfOutputGain15
    Decimation filter output gain 15
enum __pdm_data_width
    PDM data width.
    Values:
    enumerator kPDM_DataWwidth24
        PDM data width 24bit
    enumerator kPDM_DataWwidth32
        PDM data width 32bit
typedef enum __pdm_dc_removal pdm_dc_removal_t
    PDM DC remover configurations.
typedef enum __pdm_df_quality_mode pdm_df_quality_mode_t
    PDM decimation filter quality mode.
typedef enum __pdm_df_output_gain pdm_df_output_gain_t
    PDM decimation filter output gain.
typedef struct __pdm_channel_config pdm_channel_config_t
    PDM channel configurations.
typedef struct __pdm_config pdm_config_t
    PDM user configuration structure.
typedef struct __pdm_transfer pdm_transfer_t
    PDM SDMA transfer structure.
typedef struct __pdm_handle pdm_handle_t
    PDM handle.
typedef void (*pdm_transfer_callback_t)(PDM_Type *base, pdm_handle_t *handle, status_t
status, void *userData)
    PDM transfer callback prototype.
PDM_XFER_QUEUE_SIZE
    PDM XFER QUEUE SIZE.
struct __pdm_channel_config
    #include <fsl_pdm.h> PDM channel configurations.

```

Public Members

pdm_dc_remover_t outputCutOffFreq
PDM output DC remover cut off frequency

pdm_df_output_gain_t gain
Decimation Filter Output Gain

struct __pdm_config
#include <fsl_pdm.h> PDM user configuration structure.

Public Members

bool enableDoze
This module will enter disable/low leakage mode if DOZEN is active with ipg_doze is asserted

bool enableFilterBypass
Switchable bypass path for the decimation filter

uint8_t fifoWatermark
Watermark value for FIFO

pdm_df_quality_mode_t qualityMode
Quality mode

uint8_t cicOverSampleRate
CIC filter over sampling rate

struct __pdm_transfer
#include <fsl_pdm.h> PDM SDMA transfer structure.

Public Members

volatile uint8_t *data
Data start address to transfer.

volatile size_t dataSize
Total Transfer bytes size.

struct __pdm_handle
#include <fsl_pdm.h> PDM handle structure.

Public Members

uint32_t state
Transfer status

pdm_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

pdm_transfer_t pdmQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
 Index for user to queue transfer

volatile uint8_t queueDriver
 Index for driver to get the transfer data and size

uint32_t format
 data format

uint8_t watermark
 Watermark value

uint8_t startChannel
 end channel

uint8_t channelNums
 Enabled channel number

2.68 PDM EDMA Driver

void PDM_TransferInstallEDMATCDMemory(*pdm_edma_handle_t* *handle, void *tcdAddr, size_t tcdNum)

Install EDMA descriptor memory.

Parameters

- handle – Pointer to EDMA channel transfer handle.
- tcdAddr – EDMA head descriptor address.
- tcdNum – EDMA link descriptor address.

void PDM_TransferCreateHandleEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, *pdm_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the PDM Rx eDMA handle.

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

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void PDM_TransferSetMultiChannelInterleaveType(*pdm_edma_handle_t* *handle, *pdm_edma_multi_channel_interleave_t* multiChannelInterleaveType)

Initializes the multi PDM channel interleave type.

This function initializes the PDM DMA handle member interleaveType, it shall be called only when application would like to use type kPDM_EDMAMultiChannelInterleavePerChannelBlock, since the default interleaveType is kPDM_EDMAMultiChannelInterleavePerChannelSample always

Parameters

- handle – PDM eDMA handle pointer.
- multiChannelInterleaveType – Multi channel interleave type.

```
void PDM_TransferSetChannelConfigEDMA(PDM_Type *base, pdm_edma_handle_t *handle,
                                     uint32_t channel, const pdm_channel_config_t
                                     *config)
```

Configures the PDM channel.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- channel – channel index.
- config – pdm channel configurations.

```
status_t PDM_TransferReceiveEDMA(PDM_Type *base, pdm_edma_handle_t *handle,
                                 pdm_edma_transfer_t *xfer)
```

Performs a non-blocking PDM receive using eDMA.

Mcaro MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL can control whether PDM is enabled internally or externally.

- a. Scatter gather case: This function support dynamic scatter gather and static scatter gather, a. for the dynamic scatter gather case: Application should call PDM_TransferReceiveEDMA function continuously to make sure new receive request is submit before the previous one finish. b. for the static scatter gather case: Application should use the link transfer feature and make sure a loop link transfer is provided, such as:

```
pdm_edma_transfer_t pdmXfer[2] =
{
    {
        .data = s_buffer,
        .dataSize = BUFFER_SIZE,
        .linkTransfer = &pdmXfer[1],
    },

    {
        .data = &s_buffer[BUFFER_SIZE],
        .dataSize = BUFFER_SIZE,
        .linkTransfer = &pdmXfer[0]
    },
};
```

- b. Multi channel case: This function support receive multi pdm channel data, for example, if two channel is requested,

```
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_0, &channelConfig);
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_1, &channelConfig);
PDM_TransferReceiveEDMA(DEMO_PDM, &s_pdmRxHandle_0, pdmXfer);
```

The output data will be formatted as below if handle->interleaveType =

Note: This interface returns immediately after the transfer initiates. Call the PDM_GetReceiveRemainingBytes to poll the transfer status and check whether the PDM transfer is finished.

void PDM_TransferTerminateReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Terminate all PDM receive.

This function will clear all transfer slots buffered in the pdm queue. If users only want to abort the current transfer slot, please call PDM_TransferAbortReceiveEDMA.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.

void PDM_TransferAbortReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Aborts a PDM receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call PDM_TransferTerminateReceiveEDMA.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

status_t PDM_TransferGetReceiveCountEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, *size_t* *count)

Gets byte count received by PDM.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.
- count – Bytes count received by PDM.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

FSL_PDM_EDMA_DRIVER_VERSION

Version 2.6.5

enum *pdm_edma_multi_channel_interleave*

pdm multi channel interleave type

Values:

enumerator kPDM_EDMAMultiChannelInterleavePerChannelSample

enumerator kPDM_EDMAMultiChannelInterleavePerChannelBlock

typedef struct *pdm_edma_handle* *pdm_edma_handle_t*

PDM edma handler.

typedef enum *pdm_edma_multi_channel_interleave* *pdm_edma_multi_channel_interleave_t*

pdm multi channel interleave type

```
typedef struct _pdm_edma_transfer pdm_edma_transfer_t
```

PDM edma transfer.

```
typedef void (*pdm_edma_callback_t)(PDM_Type *base, pdm_edma_handle_t *handle, status_t status, void *userData)
```

PDM eDMA transfer callback function for finish and error.

MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL

the PDM enable position When calling PDM_TransferReceiveEDMA

```
struct _pdm_edma_transfer
```

```
#include <fsl_pdm_edma.h> PDM edma transfer.
```

Public Members

```
volatile uint8_t *data
```

Data start address to transfer.

```
volatile size_t dataSize
```

Total Transfer bytes size.

```
struct _pdm_edma_transfer *linkTransfer
```

linked transfer configurations

```
struct _pdm_edma_handle
```

```
#include <fsl_pdm_edma.h> PDM DMA transfer handle, users should not touch the content of the handle.
```

Public Members

```
edma_handle_t *dmaHandle
```

DMA handler for PDM send

```
uint8_t count
```

The transfer data count in a DMA request

```
uint32_t receivedBytes
```

total transfer count

```
uint32_t state
```

Internal state for PDM eDMA transfer

```
pdm_edma_callback_t callback
```

Callback for users while transfer finish or error occurs

```
bool isLoopTransfer
```

loop transfer

```
void *userData
```

User callback parameter

```
edma_tcd_t *tcd
```

TCD pool for eDMA transfer.

```
uint32_t tcdNum
```

TCD number

```
uint32_t tcdUser
```

Index for user to queue transfer.

`uint32_t tcdDriver`
Index for driver to get the transfer data and size

`volatile uint32_t tcdUsedNum`
Index for user to queue transfer.

`pdm_edma_multi_channel_interleave_t` `interleaveType`
multi channel transfer interleave type

`uint8_t endChannel`
The last enabled channel

`uint8_t channelNums`
total channel numbers

2.69 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

`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_bslice
PINT Pattern Match bit slice type.

Values:

enumerator kPINT_PatternMatchBSlice0
Bit slice 0

enum __pint_pmatch_bslice_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_bslic pint_pmatch_bslic_t
```

PINT Pattern Match bit slice type.

```
typedef enum _pint_pmatch_bslic_cfg pint_pmatch_bslic_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>
```

2.70 PORT: Port Control and Interrupts

```
static inline void PORT_GetVersionInfo(PORT_Type *base, port_version_info_t *info)
    Get PORT version information.
```

Parameters

- base – PORT peripheral base pointer
- info – PORT version information

```
static inline void PORT_SecletPortVoltageRange(PORT_Type *base, port_voltage_range_t range)
    Get PORT version information.
```

Note: : PORTA_CONFIG[RANGE] controls the voltage ranges of Port A, B, and C. Read or write PORTB_CONFIG[RANGE] and PORTC_CONFIG[RANGE] does not take effect.

Parameters

- base – PORT peripheral base pointer
- range – port voltage range

```
static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const port_pin_config_t
                                     *config)
```

Sets the port PCR register.

This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT PCR register configuration structure.

```
static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const
                                              port_pin_config_t *config)
```

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```
Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

```
static inline void PORT_SetMultipleInterruptPinsConfig(PORT_Type *base, uint32_t mask,
                                                       port_interrupt_t config)
```

Sets the port interrupt configuration in PCR register for multiple pins.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT pin interrupt configuration.
 - #kPORT_InterruptOrDMADisabled: Interrupt/DMA request disabled.
 - #kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).

- #kPORT_DMALFallingEdge: DMA request on falling edge(if the DMA requests exit).
- #kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
- #kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
- #kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
- #kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
- #kPORT_InterruptLogicZero : Interrupt when logic zero.
- #kPORT_InterruptRisingEdge : Interrupt on rising edge.
- #kPORT_InterruptFallingEdge: Interrupt on falling edge.
- #kPORT_InterruptEitherEdge : Interrupt on either edge.
- #kPORT_InterruptLogicOne : Interrupt when logic one.
- #kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
- #kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

static inline void PORT_SetPinMux(PORT_Type *base, uint32_t pin, *port_mux_t* mux)

Configures the pin muxing.

Note: : This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- mux – pin muxing slot selection.
 - kPORT_PinDisabledOrAnalog: Pin disabled or work in analog function.
 - kPORT_MuxAsGpio : Set as GPIO.
 - kPORT_MuxAlt2 : chip-specific.
 - kPORT_MuxAlt3 : chip-specific.
 - kPORT_MuxAlt4 : chip-specific.
 - kPORT_MuxAlt5 : chip-specific.
 - kPORT_MuxAlt6 : chip-specific.
 - kPORT_MuxAlt7 : chip-specific.

static inline void PORT_EnablePinsDigitalFilter(PORT_Type *base, uint32_t mask, bool enable)

Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

```
static inline void PORT_SetDigitalFilterConfig(PORT_Type *base, const
                                             port_digital_filter_config_t *config)
```

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

```
static inline void PORT_SetPinDriveStrength(PORT_Type *base, uint32_t pin, uint8_t strength)
```

Configures the port pin drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- strength – PORT pin drive strength
 - kPORT_LowDriveStrength = 0U - Low-drive strength is configured.
 - kPORT_HighDriveStrength = 1U - High-drive strength is configured.

```
static inline void PORT_EnablePinDoubleDriveStrength(PORT_Type *base, uint32_t pin, bool
                                                    enable)
```

Enables the port pin double drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- enable – PORT pin drive strength configuration.

```
static inline void PORT_SetPinPullValue(PORT_Type *base, uint32_t pin, uint8_t value)
```

Configures the port pin pull value.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- value – PORT pin pull value
 - kPORT_LowPullResistor = 0U - Low internal pull resistor value is selected.
 - kPORT_HighPullResistor = 1U - High internal pull resistor value is selected.

```
static inline uint32_t PORT_GetEFTDetectFlags(PORT_Type *base)
```

Get EFT detect flags.

Parameters

- base – PORT peripheral base pointer

Returns

EFT detect flags

```
static inline void PORT_EnableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
```

Enable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_DisableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
Disable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_ClearAllLowEFTDetectors(PORT_Type *base)
Clear all low EFT detector.

Note: : Port B and Port C pins share the same EFT detector clear control from PORTC_EDCR register. Any write to the PORTB_EDCR does not take effect.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_ClearAllHighEFTDetectors(PORT_Type *base)
Clear all high EFT detector.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

FSL_PORT_DRIVER_VERSION
PORT driver version.

enum __port_pull
Internal resistor pull feature selection.

Values:

enumerator kPORT_PullDisable
Internal pull-up/down resistor is disabled.

enumerator kPORT_PullDown
Internal pull-down resistor is enabled.

enumerator kPORT_PullUp
Internal pull-up resistor is enabled.

enum __port_pull_value
Internal resistor pull value selection.

Values:

enumerator kPORT_LowPullResistor
Low internal pull resistor value is selected.

enumerator kPORT_HighPullResistor
High internal pull resistor value is selected.

enum __port_slew_rate
Slew rate selection.

Values:

enumerator kPORT_FastSlewRate
Fast slew rate is configured.

enumerator kPORT_SlowSlewRate
Slow slew rate is configured.

enum __port_open_drain_enable
Open Drain feature enable/disable.

Values:

enumerator kPORT_OpenDrainDisable
Open drain output is disabled.

enumerator kPORT_OpenDrainEnable
Open drain output is enabled.

enum __port_passive_filter_enable
Passive filter feature enable/disable.

Values:

enumerator kPORT_PassiveFilterDisable
Passive input filter is disabled.

enumerator kPORT_PassiveFilterEnable
Passive input filter is enabled.

enum __port_drive_strength
Configures the drive strength.

Values:

enumerator kPORT_LowDriveStrength
Low-drive strength is configured.

enumerator kPORT_HighDriveStrength
High-drive strength is configured.

enum __port_drive_strength1
Configures the drive strength1.

Values:

enumerator kPORT_NormalDriveStrength
Normal drive strength

enumerator kPORT_DoubleDriveStrength
Double drive strength

enum __port_input_buffer
input buffer disable/enable.

Values:

enumerator kPORT_InputBufferDisable
Digital input is disabled

enumerator kPORT_InputBufferEnable
Digital input is enabled

enum __port_invert_input
Digital input is not inverted or it is inverted.

Values:

enumerator kPORT_InputNormal
Digital input is not inverted

enumerator kPORT_InputInvert

Digital input is inverted

enum __port_lock_register

Unlock/lock the pin control register field[15:0].

Values:

enumerator kPORT_UnlockRegister

Pin Control Register fields [15:0] are not locked.

enumerator kPORT_LockRegister

Pin Control Register fields [15:0] are locked.

enum __port_mux

Pin mux selection.

Values:

enumerator kPORT_PinDisabledOrAnalog

Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT_MuxAsGpio

Corresponding pin is configured as GPIO.

enumerator kPORT_MuxAlt0

Chip-specific

enumerator kPORT_MuxAlt1

Chip-specific

enumerator kPORT_MuxAlt2

Chip-specific

enumerator kPORT_MuxAlt3

Chip-specific

enumerator kPORT_MuxAlt4

Chip-specific

enumerator kPORT_MuxAlt5

Chip-specific

enumerator kPORT_MuxAlt6

Chip-specific

enumerator kPORT_MuxAlt7

Chip-specific

enumerator kPORT_MuxAlt8

Chip-specific

enumerator kPORT_MuxAlt9

Chip-specific

enumerator kPORT_MuxAlt10

Chip-specific

enumerator kPORT_MuxAlt11

Chip-specific

enumerator kPORT_MuxAlt12

Chip-specific

enumerator kPORT_MuxAlt13
Chip-specific

enumerator kPORT_MuxAlt14
Chip-specific

enumerator kPORT_MuxAlt15
Chip-specific

enum _port_digital_filter_clock_source
Digital filter clock source selection.

Values:

enumerator kPORT_BusClock
Digital filters are clocked by the bus clock.

enumerator kPORT_LpoClock
Digital filters are clocked by the 1 kHz LPO clock.

enum _port_voltage_range
PORT voltage range.

Values:

enumerator kPORT_VoltageRange1Dot71V_3Dot6V
Port voltage range is 1.71 V - 3.6 V.

enumerator kPORT_VoltageRange2Dot70V_3Dot6V
Port voltage range is 2.70 V - 3.6 V.

typedef enum _port_mux port_mux_t
Pin mux selection.

typedef enum _port_digital_filter_clock_source port_digital_filter_clock_source_t
Digital filter clock source selection.

typedef struct _port_digital_filter_config port_digital_filter_config_t
PORT digital filter feature configuration definition.

typedef struct _port_pin_config port_pin_config_t
PORT pin configuration structure.

typedef struct _port_version_info port_version_info_t
PORT version information.

typedef enum _port_voltage_range port_voltage_range_t
PORT voltage range.

FSL_COMPONENT_ID

struct _port_digital_filter_config
#include <fsl_port.h> PORT digital filter feature configuration definition.

Public Members

uint32_t digitalFilterWidth
Set digital filter width

port_digital_filter_clock_source_t clockSource
Set digital filter clockSource

struct _port_pin_config
#include <fsl_port.h> PORT pin configuration structure.

Public Members

```

uint16_t pullSelect
    No-pull/pull-down/pull-up select
uint16_t pullValueSelect
    Pull value select
uint16_t slewRate
    Fast/slow slew rate Configure
uint16_t passiveFilterEnable
    Passive filter enable/disable
uint16_t openDrainEnable
    Open drain enable/disable
uint16_t driveStrength
    Fast/slow drive strength configure
uint16_t driveStrength1
    Normal/Double drive strength enable/disable
uint16_t inputBuffer
    Input Buffer Configure
uint16_t invertInput
    Invert Input Configure
uint16_t lockRegister
    Lock/unlock the PCR field[15:0]
struct __port__version__info
    #include <fsl_port.h> PORT version information.

```

Public Members

```

uint16_t feature
    Feature Specification Number.
uint8_t minor
    Minor Version Number.
uint8_t major
    Major Version Number.

```

2.71 PUF: Physical Unclonable Function

```

FSL_PUF_DRIVER_VERSION
    PUF driver version. Version 2.2.0.
    Current version: 2.2.0
    Change log:
        • 2.0.0
            – Initial version.
        • 2.0.1

```

- Fixed puf_wait_usec function optimization issue.
- 2.0.2
 - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
 - Fix MISRA C-2012 issue.
- 2.1.0
 - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
 - Update initialization logic .
- 2.1.1
 - Fix ARMGCC build warning .
- 2.1.2
 - Update: Add automatic big to little endian swap for user (pre-shared) keys destined to secret hardware bus (PUF key index 0).
- 2.1.3
 - Fix MISRA C-2012 issue.
- 2.1.4
 - Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.
- 2.1.5
 - Use common SDK delay in puf_wait_usec()
- 2.1.6
 - Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.
- 2.2.0
- Add support for kPUF_KeySlot4.
- Add new PUF_ClearKey() function, that clears a desired PUF internal HW key register.

enum __puf_key_index_register

Values:

enumerator kPUF_KeyIndex_00

enumerator kPUF_KeyIndex_01

enumerator kPUF_KeyIndex_02

enumerator kPUF_KeyIndex_03

enumerator kPUF_KeyIndex_04

enumerator kPUF_KeyIndex_05

enumerator kPUF_KeyIndex_06

enumerator kPUF_KeyIndex_07

enumerator kPUF_KeyIndex_08

```

    enumerator kPUF__KeyIndex__09
    enumerator kPUF__KeyIndex__10
    enumerator kPUF__KeyIndex__11
    enumerator kPUF__KeyIndex__12
    enumerator kPUF__KeyIndex__13
    enumerator kPUF__KeyIndex__14
    enumerator kPUF__KeyIndex__15

enum _puf_min_max
    Values:
    enumerator kPUF__KeySizeMin
    enumerator kPUF__KeySizeMax
    enumerator kPUF__KeyIndexMax

enum _puf_key_slot
    PUF key slot.
    Values:
    enumerator kPUF__KeySlot0
        PUF key slot 0
    enumerator kPUF__KeySlot1
        PUF key slot 1

    PUF status return codes.
    Values:
    enumerator kStatus__EnrollNotAllowed
    enumerator kStatus__StartNotAllowed

typedef enum _puf_key_index_register puf_key_index_register_t
typedef enum _puf_min_max puf_min_max_t
typedef enum _puf_key_slot puf_key_slot_t
    PUF key slot.

PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x)
    Get Key Code size in bytes from key size in bytes at compile time.
PUF_MIN_KEY_CODE_SIZE
PUF_ACTIVATION_CODE_SIZE
KEYSTORE_PUF_DISCHARGE_TIME_FIRST_TRY_MS
KEYSTORE_PUF_DISCHARGE_TIME_MAX_MS

struct puf_config_t
    #include <fsl_puf.h>

```

2.72 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.h>
struct puf_key_ctx_t
 #include <fsl_puf_v3.h>

2.73 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 = $f_{clk}/1$

enumerator `kPWM_Prescale_Divide_2`

PWM clock frequency = $f_{clk}/2$

enumerator `kPWM_Prescale_Divide_4`

PWM clock frequency = $f_{clk}/4$

enumerator `kPWM_Prescale_Divide_8`

PWM clock frequency = $f_{clk}/8$

enumerator `kPWM_Prescale_Divide_16`

PWM clock frequency = $f_{clk}/16$

enumerator `kPWM_Prescale_Divide_32`

PWM clock frequency = $f_{clk}/32$

enumerator `kPWM_Prescale_Divide_64`

PWM clock frequency = $f_{clk}/64$

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_LoadEvery8Oportunity

Every 8 PWM opportunities

enumerator kPWM_LoadEvery9Oportunity

Every 9 PWM opportunities

enumerator kPWM_LoadEvery10Oportunity

Every 10 PWM opportunities

enumerator kPWM_LoadEvery11Oportunity

Every 11 PWM opportunities

enumerator kPWM_LoadEvery12Oportunity

Every 12 PWM opportunities

enumerator kPWM_LoadEvery13Oportunity

Every 13 PWM opportunities

enumerator kPWM_LoadEvery14Oportunity

Every 14 PWM opportunities

enumerator kPWM_LoadEvery15Oportunity

Every 15 PWM opportunities

enumerator kPWM_LoadEvery16Oportunity

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 center-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 buffered, the new value will not active unless call `PWM_SetPwmLdok()` and the reload point is reached.

Parameters

- `base` – PWM peripheral base address
- `subModule` – PWM submodule to configure
- `valueRegister` – VALx register that will be written new value
- `value` – Value that will be written 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

2.74 QDC: Quadrature Encoder/Decoder

void QDC_Init(QDC_Type *base, const qdc_config_t *config)

Initialization for the QDC module.

This function is to make the initialization for the QDC module. It should be called firstly before any operation to the QDC with the operations like:

- Enable the clock for QDC module.

- Configure the QDC's working attributes.

Parameters

- base – QDC peripheral base address.
- config – Pointer to configuration structure. See to “qdc_config_t”.

void QDC_Deinit(QDC_Type *base)

De-initialization for the QDC module.

This function is to make the de-initialization for the QDC module. It could be called when QDC is no longer used with the operations like:

- Disable the clock for QDC module.

Parameters

- base – QDC peripheral base address.

void QDC_GetDefaultConfig(*qdc_config_t* *config)

Get an available pre-defined settings for QDC's configuration.

This function initializes the QDC configuration structure with an available settings, the default value are:

```
config->enableReverseDirection      = false;
config->decoderWorkMode             = kQDC_DecoderWorkAsNormalMode;
config->HOMETriggerMode             = kQDC_HOMETriggerDisabled;
config->INDEXTriggerMode            = kQDC_INDEXTriggerDisabled;
config->enableTRIGGERClearPositionCounter = false;
config->enableTRIGGERClearHoldPositionCounter = false;
config->enableWatchdog              = false;
config->watchdogTimeoutValue         = 0U;
config->filterCount                  = 0U;
config->filterSamplePeriod           = 0U;
config->positionMatchMode            = kQDC_
↪ POSMATCHOnPositionCounterEqualToComapreValue;
config->positionCompareValue         = 0xFFFFFFFFU;
config->revolutionCountCondition     = kQDC_RevolutionCountOnINDEXPulse;
config->enableModuloCountMode        = false;
config->positionModulusValue         = 0U;
config->positionInitialValue         = 0U;
config->prescalerValue               = kQDC_ClockDiv1;
config->enablePeriodMeasurementFunction = true;
```

Parameters

- config – Pointer to a variable of configuration structure. See to “qdc_config_t”.

void QDC_DoSoftwareLoadInitialPositionValue(QDC_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 – QDC peripheral base address.

void QDC_SetSelfTestConfig(QDC_Type *base, const *qdc_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 – QDC peripheral base address.
- config – Pointer to configuration structure. See to “qdc_self_test_config_t”. Pass “NULL” to disable.

void QDC_EnableWatchdog(QDC_Type *base, bool enable)

Enable watchdog for QDC module.

Parameters

- base – QDC peripheral base address
- enable – Enables or disables the watchdog

void QDC_SetInitialPositionValue(QDC_Type *base, uint32_t value)

Set initial position value for QDC module.

Parameters

- base – QDC peripheral base address
- value – Positive initial value

uint32_t QDC_GetStatusFlags(QDC_Type *base)

Get the status flags.

Parameters

- base – QDC peripheral base address.

Returns

Mask value of status flags. For available mask, see to “_qdc_status_flags”.

void QDC_ClearStatusFlags(QDC_Type *base, uint32_t mask)

Clear the status flags.

Parameters

- base – QDC peripheral base address.
- mask – Mask value of status flags to be cleared. For available mask, see to “_qdc_status_flags”.

static inline uint16_t QDC_GetSignalStatusFlags(QDC_Type *base)

Get the signals' real-time status.

Parameters

- base – QDC peripheral base address.

Returns

Mask value of signals' real-time status. For available mask, see to “_qdc_signal_status_flags”

void QDC_EnableInterrupts(QDC_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – QDC peripheral base address.
- mask – Mask value of interrupts to be enabled. For available mask, see to “_qdc_interrupt_enable”.

`void QDC_DisableInterrupts(QDC_Type *base, uint32_t mask)`

Disable the interrupts.

Parameters

- `base` – QDC peripheral base address.
- `mask` – Mask value of interrupts to be disabled. For available mask, see to “_qdc_interrupt_enable”.

`uint32_t QDC_GetEnabledInterrupts(QDC_Type *base)`

Get the enabled interrupts' flags.

Parameters

- `base` – QDC peripheral base address.

Returns

Mask value of enabled interrupts.

`uint32_t QDC_GetPositionValue(QDC_Type *base)`

Get the current position counter's value.

Parameters

- `base` – QDC peripheral base address.

Returns

Current position counter's value.

`uint32_t QDC_GetHoldPositionValue(QDC_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` – QDC peripheral base address.

Returns

Hold position counter's value.

`static inline uint16_t QDC_GetPositionDifferenceValue(QDC_Type *base)`

Get the position difference counter's value.

Parameters

- `base` – QDC peripheral base address.

Returns

The position difference counter's value.

`static inline uint16_t QDC_GetHoldPositionDifferenceValue(QDC_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` – QDC peripheral base address.

Returns

Hold position difference counter's value.

```
static inline uint16_t QDC_GetRevolutionValue(QDC_Type *base)
```

Get the position revolution counter's value.

Parameters

- base – QDC peripheral base address.

Returns

The position revolution counter's value.

```
static inline uint16_t QDC_GetHoldRevolutionValue(QDC_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 – QDC peripheral base address.

Returns

Hold position revolution counter's value.

```
static inline uint16_t QDC_GetLastEdgeTimeValue(QDC_Type *base)
```

Get the last edge time value.

Parameters

- base – QDC peripheral base address.

Returns

The last edge time hold value.

```
static inline uint16_t QDC_GetHoldLastEdgeTimeValue(QDC_Type *base)
```

Get the last edge time hold value.

Parameters

- base – QDC peripheral base address.

Returns

The last edge time hold value.

```
static inline uint16_t QDC_GetPositionDifferencePeriodValue(QDC_Type *base)
```

Get the position difference period value.

Parameters

- base – QDC peripheral base address.

Returns

The position difference period hold value.

```
static inline uint16_t QDC_GetPositionDifferencePeriodBufferValue(QDC_Type *base)
```

Get the position difference period buffer value.

Parameters

- base – QDC peripheral base address.

Returns

The position difference period hold value.

```
static inline uint16_t QDC_GetHoldPositionDifferencePeriodValue(QDC_Type *base)
```

Get the position difference period hold value.

Parameters

- base – QDC peripheral base address.

Returns

The position difference period hold value.

enum `_qdc_interrupt_enable`

Interrupt enable/disable mask.

Values:

enumerator `kQDC_HOMETransitionInterruptEnable`

HOME interrupt enable.

enumerator `kQDC_INDEXPulseInterruptEnable`

INDEX pulse interrupt enable.

enumerator `kQDC_WatchdogTimeoutInterruptEnable`

Watchdog timeout interrupt enable.

enumerator `kQDC_PositionCompareInterruptEnable`

Position compare interrupt enable.

enumerator `kQDC_PositionRollOverInterruptEnable`

Roll-over interrupt enable.

enumerator `kQDC_PositionRollUnderInterruptEnable`

Roll-under interrupt enable.

enum `_qdc_status_flags`

Status flag mask.

These flags indicate the counter's events.

Values:

enumerator `kQDC_HOMETransitionFlag`

HOME signal transition interrupt request.

enumerator `kQDC_INDEXPulseFlag`

INDEX Pulse Interrupt Request.

enumerator `kQDC_WatchdogTimeoutFlag`

Watchdog timeout interrupt request.

enumerator `kQDC_PositionCompareFlag`

Position compare interrupt request.

enumerator `kQDC_PositionRollOverFlag`

Roll-over interrupt request.

enumerator `kQDC_PositionRollUnderFlag`

Roll-under interrupt request.

enumerator `kQDC_LastCountDirectionFlag`

Last count was in the up direction, or the down direction.

enum `_qdc_signal_status_flags`

Signal status flag mask.

These flags indicate the counter's signal.

Values:

enumerator `kQDC_RawHOMESTatusFlag`

Raw HOME input.

enumerator kQDC_RawINDEXStatusFlag

Raw INDEX input.

enumerator kQDC_RawPHBStatusFlag

Raw PHASEB input.

enumerator kQDC_RawPHAEXStatusFlag

Raw PHASEA input.

enumerator kQDC_FilteredHOMESTatusFlag

The filtered version of HOME input.

enumerator kQDC_FilteredINDEXStatusFlag

The filtered version of INDEX input.

enumerator kQDC_FilteredPHBStatusFlag

The filtered version of PHASEB input.

enumerator kQDC_FilteredPHASTatusFlag

The filtered version of PHASEA input.

enum _qdc_home_trigger_mode

Define HOME signal's trigger mode.

The QDC would count the trigger from HOME signal line.

Values:

enumerator kQDC_HOMETriggerDisabled

HOME signal's trigger is disabled.

enumerator kQDC_HOMETriggerOnRisingEdge

Use positive going edge-to-trigger initialization of position counters.

enumerator kQDC_HOMETriggerOnFallingEdge

Use negative going edge-to-trigger initialization of position counters.

enum _qdc_index_trigger_mode

Define INDEX signal's trigger mode.

The QDC would count the trigger from INDEX signal line.

Values:

enumerator kQDC_INDEXTriggerDisabled

INDEX signal's trigger is disabled.

enumerator kQDC_INDEXTriggerOnRisingEdge

Use positive going edge-to-trigger initialization of position counters.

enumerator kQDC_INDEXTriggerOnFallingEdge

Use negative going edge-to-trigger initialization of position counters.

enum _qdc_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 kQDC_DecoderWorkAsNormalMode

Use standard quadrature decoder with PHASEA and PHASEB.

enumerator kQDC_DecoderWorkAsSignalPhaseCountMode

PHASEA input generates a count signal while PHASEB input control the direction.

enum __qdc_position_match_mode

Define type for the condition of POSMATCH pulses.

Values:

enumerator kQDC_POSMATCHOnPositionCounterEqualToCompareValue

POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

enumerator kQDC_POSMATCHOnReadingAnyPositionCounter

POSMATCH pulses when any position counter register is read.

enum __qdc_revolution_count_condition

Define type for determining how the revolution counter (REV) is incremented/decremented.

Values:

enumerator kQDC_RevolutionCountOnINDEXPulse

Use INDEX pulse to increment/decrement revolution counter.

enumerator kQDC_RevolutionCountOnRollOverModulus

Use modulus counting roll-over/under to increment/decrement revolution counter.

enum __qdc_self_test_direction

Define type for direction of self test generated signal.

Values:

enumerator kQDC_SelfTestDirectionPositive

Self test generates the signal in positive direction.

enumerator kQDC_SelfTestDirectionNegative

Self test generates the signal in negative direction.

enum __qdc_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 kQDC_ClockDiv1

enumerator kQDC_ClockDiv2

enumerator kQDC_ClockDiv4

enumerator kQDC_ClockDiv8

enumerator kQDC_ClockDiv16

enumerator kQDC_ClockDiv32

enumerator kQDC_ClockDiv64

enumerator kQDC_ClockDiv128

enumerator kQDC_ClockDiv256
enumerator kQDC_ClockDiv512
enumerator kQDC_ClockDiv1024
enumerator kQDC_ClockDiv2048
enumerator kQDC_ClockDiv4096
enumerator kQDC_ClockDiv8192
enumerator kQDC_ClockDiv16384
enumerator kQDC_ClockDiv32768

enum _qdc_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 kQDC_FilterPrescalerDiv1
Input filter prescaler is 1.
enumerator kQDC_FilterPrescalerDiv2
Input filter prescaler is 2.
enumerator kQDC_FilterPrescalerDiv4
Input filter prescaler is 4.
enumerator kQDC_FilterPrescalerDiv8
Input filter prescaler is 8.
enumerator kQDC_FilterPrescalerDiv16
Input filter prescaler is 16.
enumerator kQDC_FilterPrescalerDiv32
Input filter prescaler is 32.
enumerator kQDC_FilterPrescalerDiv64
Input filter prescaler is 64.
enumerator kQDC_FilterPrescalerDiv128
Input filter prescaler is 128.

typedef enum _qdc_home_trigger_mode qdc_home_trigger_mode_t

Define HOME signal's trigger mode.

The QDC would count the trigger from HOME signal line.

typedef enum _qdc_index_trigger_mode qdc_index_trigger_mode_t

Define INDEX signal's trigger mode.

The QDC would count the trigger from INDEX signal line.

typedef enum _qdc_decoder_work_mode qdc_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 _qdc_position_match_mode qdc_position_match_mode_t
```

Define type for the condition of POSMATCH pulses.

```
typedef enum _qdc_revolution_count_condition qdc_revolution_count_condition_t
```

Define type for determining how the revolution counter (REV) is incremented/decremented.

```
typedef enum _qdc_self_test_direction qdc_self_test_direction_t
```

Define type for direction of self test generated signal.

```
typedef enum _qdc_prescaler qdc_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 _qdc_filter_prescaler qdc_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 _qdc_config qdc_config_t
```

Define user configuration structure for QDC module.

```
typedef struct _qdc_self_test_config qdc_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_QDC_DRIVER_VERSION
```

```
struct _qdc_config
```

#include <fsl_qdc.h> Define user configuration structure for QDC module.

Public Members

```
bool enableReverseDirection
```

Enable reverse direction counting.

```
qdc_decoder_work_mode_t decoderWorkMode
```

Enable signal phase count mode.

```
qdc_home_trigger_mode_t HOMETriggerMode
```

Enable HOME to initialize position counters.

```
qdc_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.

qdc_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.

qdc_position_match_mode_t positionMatchMode

The condition of POSMATCH pulses.

uint32_t positionCompareValue

Position compare value. The available value is a 32-bit number.

qdc_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.

qdc_prescaler_t prescalerValue

The value of prescaler.

struct __qdc_self_test_config

#include <fsl_qdc.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

qdc_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.

2.75 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 kRST_None

None RESET gate.

enumerator kFMU_RST_SHIFT_RSTn

Flash management unit reset control

enumerator kMUX_RST_SHIFT_RSTn

Input mux reset control

enumerator kPORT0_RST_SHIFT_RSTn

PORT0 reset control

enumerator kPORT1_RST_SHIFT_RSTn

PORT1 reset control

enumerator kPORT2_RST_SHIFT_RSTn

PORT2 reset control

enumerator kPORT3_RST_SHIFT_RSTn

PORT3 reset control

enumerator kPORT4_RST_SHIFT_RSTn

PORT4 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 kGPIO4_RST_SHIFT_RSTn

GPIO4 reset control

enumerator kPINT_RST_SHIFT_RSTn

Pin interrupt (PINT) reset control

enumerator kDMA0_RST_SHIFT_RSTn

DMA0 reset control

enumerator kCRC_RST_SHIFT_RSTn

CRC reset control

enumerator kMRT_RST_SHIFT_RSTn
Multi-rate timer (MRT) reset control

enumerator kOSTIMER_RST_SHIFT_RSTn
OSTimer reset control

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control

enumerator kADC1_RST_SHIFT_RSTn
ADC1 reset control

enumerator kUTICK_RST_SHIFT_RSTn
Micro-tick timer reset control

enumerator kFC0_RST_SHIFT_RSTn
Flexcomm Interface 0 reset control

enumerator kFC1_RST_SHIFT_RSTn
Flexcomm Interface 1 reset control

enumerator kFC2_RST_SHIFT_RSTn
Flexcomm Interface 2 reset control

enumerator kFC3_RST_SHIFT_RSTn
Flexcomm Interface 3 reset control

enumerator kFC4_RST_SHIFT_RSTn
Flexcomm Interface 4 reset control

enumerator kFC5_RST_SHIFT_RSTn
Flexcomm Interface 5 reset control

enumerator kFC6_RST_SHIFT_RSTn
Flexcomm Interface 6 reset control

enumerator kFC7_RST_SHIFT_RSTn
Flexcomm Interface 7 reset control

enumerator kMICFIL_RST_SHIFT_RSTn
Flexcomm Interface 7 reset control

enumerator kCTIMER2_RST_SHIFT_RSTn
CTimer 2 reset control

enumerator kUSB0_FS_DCD_RST_SHIFT_RSTn
USB0-FS DCD reset control

enumerator kCTIMER0_RST_SHIFT_RSTn
CTimer 0 reset control

enumerator kCTIMER1_RST_SHIFT_RSTn
CTimer 1 reset control

enumerator kSMART_DMA_RST_SHIFT_RSTn
SmartDMA reset control

enumerator kDMA1_RST_SHIFT_RSTn
DMA1 reset control

enumerator kUSDHC_RST_SHIFT_RSTn
uSDHC reset control

enumerator kFLEXIO_RST_SHIFT_RSTn
FLEXIO reset control

enumerator kSAI0_RST_SHIFT_RSTn
SAI0 reset control

enumerator kSAI1_RST_SHIFT_RSTn
SAI1 reset control

enumerator kTRO_RST_SHIFT_RSTn
TRO reset control

enumerator kFREQME_RST_SHIFT_RSTn
FREQME reset control

enumerator kTRNG_RST_SHIFT_RSTn
TRNG reset control

enumerator kFLEXCAN0_RST_SHIFT_RSTn
Flexcan0 reset control

enumerator kFLEXCAN1_RST_SHIFT_RSTn
Flexcan1 reset control

enumerator kUSB_HS_RST_SHIFT_RSTn
USB HS reset control

enumerator kUSB_HS_PHY_RST_SHIFT_RSTn
USB HS PHY reset control

enumerator kCTIMER3_RST_SHIFT_RSTn
CTimer 3 reset control

enumerator kCTIMER4_RST_SHIFT_RSTn
CTimer 4 reset control

enumerator kPUF_RST_SHIFT_RSTn
PUF reset control

enumerator kPKC_RST_SHIFT_RSTn
PKC reset control

enumerator kSM3_RST_SHIFT_RSTn
SM3 reset control

enumerator kI3C0_RST_SHIFT_RSTn
I3C0 reset control

enumerator kI3C1_RST_SHIFT_RSTn
I3C1 reset control

enumerator kQDC0_RST_SHIFT_RSTn
QDC0 reset control

enumerator kQDC1_RST_SHIFT_RSTn
QDC1 reset control

enumerator kPWM0_RST_SHIFT_RSTn
PWM0 reset control

enumerator kPWM1_RST_SHIFT_RSTn
PWM1 reset control

enumerator kAOIO_RST_SHIFT_RSTn
AOIO reset control

enumerator kVREF_RST_SHIFT_RSTn
VREF reset control

enumerator kEWM_RST_SHIFT_RSTn
EWM reset control

enumerator kEIM_RST_SHIFT_RSTn
EIM 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

CRC_RSTS

CTIMER_RSTS
DMA_RSTS_N
LP_FLEXCOMM_RSTS
GPIO_RSTS_N
INPUTMUX_RSTS
FLASH_RSTS
MRT_RSTS
PINT_RSTS
TRNG_RSTS
UTICK_RSTS
OSTIMER_RSTS
I3C_RSTS

2.76 RTC: Real Time Clock

`void RTC_Init(RTC_Type *base, const rtc_config_t *config)`

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

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

Parameters

- base – RTC peripheral base address
- config – Pointer to the user's RTC configuration structure.

`void RTC_Deinit(RTC_Type *base)`

Stops the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

`void RTC_GetDefaultConfig(rtc_config_t *config)`

Fills in the RTC config struct with the default settings.

The default values are as follows.

```
config->clockSource = kRTC_BusClock;  
config->prescaler = kRTC_ClockDivide_16_2048;  
config->time_us = 1000000U;
```

Parameters

- config – Pointer to the user's RTC configuration structure.

status_t RTC_SetDatetime(*rtc_datetime_t* *datetime)

Sets the RTC date and time according to the given time structure.

Parameters

- *datetime* – Pointer to the structure where the date and time details 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_datetime_t* *datetime)

Gets the RTC time and stores it in the given time structure.

Parameters

- *datetime* – Pointer to the structure where the date and time details are stored.

void RTC_SetAlarm(*uint32_t* second)

Sets the RTC alarm time.

Parameters

- *second* – Second value. User input the number of second. After seconds user input, alarm occurs.

void RTC_GetAlarm(*rtc_datetime_t* *datetime)

Returns the RTC alarm time.

Parameters

- *datetime* – Pointer to the structure where the alarm date and time details are stored.

void RTC_SetAlarmCallback(*rtc_alarm_callback_t* callback)

Set the RTC alarm callback.

Parameters

- *callback* – The callback function.

static inline void RTC_SelectSourceClock(*RTC_Type* *base, *rtc_clock_source_t* clock,
rtc_clock_prescaler_t divide)

Select Real-Time Clock Source and Clock Prescaler.

Parameters

- *base* – RTC peripheral base address
- *clock* – Select RTC clock source
- *divide* – Select RTC clock prescaler value

uint32_t RTC_GetDivideValue(*RTC_Type* *base)

Get the RTC Divide value.

Note: This API should be called after selecting clock source and clock prescaler.

Parameters

- *base* – RTC peripheral base address

Returns

The Divider value. The Divider value depends on clock source and clock prescaler

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. This is a logical OR of members of the enumeration `rtc_interrupt_enable_t`

static inline void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.

Parameters

- base – PIT peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration `rtc_interrupt_enable_t`

static inline uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled RTC interrupts.

Parameters

- base – RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration `rtc_interrupt_enable_t`

static inline uint32_t RTC_GetInterruptFlags(RTC_Type *base)

Gets the RTC interrupt flags.

Parameters

- base – RTC peripheral base address

Returns

The interrupt flags. This is the logical OR of members of the enumeration `rtc_interrupt_flags_t`

static inline void RTC_ClearInterruptFlags(RTC_Type *base, uint32_t mask)

Clears the RTC interrupt flags.

Parameters

- base – RTC peripheral base address
- mask – The interrupt flags to clear. This is a logical OR of members of the enumeration `rtc_interrupt_flags_t`

static inline void RTC_EnableOutput(RTC_Type *base, uint32_t mask)

Enable the RTC output. If RTC output is enabled, the RTCO pinout will be toggled when RTC counter overflows.

Parameters

- base – RTC peripheral base address
- mask – The Output to enable. This is a logical OR of members of the enumeration `rtc_output_enable_t`

static inline void RTC_DisableOutput(RTC_Type *base, uint32_t mask)

Disable the RTC output.

Parameters

- base – RTC peripheral base address

- **mask** – The Output to disable. This is a logical OR of members of the enumeration `rtc_output_enable_t`

static inline void RTC_SetModuloValue(RTC_Type *base, uint32_t value)

Set the RTC module value.

Parameters

- **base** – RTC peripheral base address
- **value** – The Module Value. The RTC Modulo register allows the compare value to be set to any value from 0x0000 to 0xFFFF

static inline uint16_t RTC_GetCountValue(RTC_Type *base)

Get the RTC Count value.

Parameters

- **base** – RTC peripheral base address

Returns

The Count Value. The Count Value is allowed from 0x0000 to 0xFFFF

FSL_RTC_DRIVER_VERSION

Version 2.0.6

enum _rtc_clock_source

List of RTC clock source.

Values:

enumerator kRTC_ExternalClock

External clock source

enumerator kRTC_LPOCLK

Real-time clock source is 1 kHz (LPOCLK)

enumerator kRTC_ICSIRCLK

Internal reference clock (ICSIRCLK)

enumerator kRTC_BusClock

Bus clock

enum _rtc_clock_prescaler

List of RTC clock prescaler.

Values:

enumerator kRTC_ClockDivide_off

Off

enumerator kRTC_ClockDivide_1_128

If RTCLKS = x0, it is 1; if RTCLKS = x1, it is 128

enumerator kRTC_ClockDivide_2_256

If RTCLKS = x0, it is 2; if RTCLKS = x1, it is 256

enumerator kRTC_ClockDivide_4_512

If RTCLKS = x0, it is 4; if RTCLKS = x1, it is 512

enumerator kRTC_ClockDivide_8_1024

If RTCLKS = x0, it is 8; if RTCLKS = x1, it is 1024

enumerator kRTC_ClockDivide_16_2048

If RTCLKS = x0, it is 16; if RTCLKS = x1, it is 2048

enumerator kRTC_ClockDivide_32_100

If RTCLKS = x0, it is 32; if RTCLKS = x1, it is 100

enumerator kRTC_ClockDivide_64_1000

If RTCLKS = x0, it is 64; if RTCLKS = x1, it is 1000

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_InterruptEnable

Interrupt enable

enum _RTC_interrupt_flags

List of RTC Interrupt flags.

Values:

enumerator kRTC_InterruptFlag

Interrupt flag

enum _RTC_output_enable

List of RTC Output.

Values:

enumerator kRTC_OutputEnable

Output enable

typedef struct _rtc_datetime rtc_datetime_t

Structure is used to hold the date and time.

typedef enum _rtc_clock_source rtc_clock_source_t

List of RTC clock source.

typedef enum _rtc_clock_prescaler rtc_clock_prescaler_t

List of RTC clock prescaler.

typedef enum _rtc_interrupt_enable rtc_interrupt_enable_t

List of RTC interrupts.

typedef enum _RTC_interrupt_flags rtc_interrupt_flags_t

List of RTC Interrupt flags.

typedef enum _RTC_output_enable rtc_output_enable_t

List of RTC Output.

typedef struct _rtc_config rtc_config_t

RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the RTC_GetDefaultConfig() function and pass a pointer to your config structure instance.

typedef void (*rtc_alarm_callback_t)(void)

RTC alarm callback function.

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> RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the `RTC_GetDefaultConfig()` function and pass a pointer to your config structure instance.

2.77 SAI: Serial Audio Interface

2.78 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless `SAI_Init` is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless `SAI_TxInit` or `SAI_RxInit` is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

`void SAI_RxReset(I2S_Type *base)`

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- `base` – SAI base pointer

`void SAI_TxEnable(I2S_Type *base, bool enable)`

Enables/disables the SAI Tx.

Parameters

- `base` – SAI base pointer.
- `enable` – True means enable SAI Tx, false means disable.

`void SAI_RxEnable(I2S_Type *base, bool enable)`

Enables/disables the SAI Rx.

Parameters

- `base` – SAI base pointer.
- `enable` – True means enable SAI Rx, false means disable.

`static inline void SAI_TxSetBitClockDirection(I2S_Type *base, sai_master_slave_t masterSlave)`

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- `base` – SAI base pointer.
- `masterSlave` – reference `sai_master_slave_t`.

`static inline void SAI_RxSetBitClockDirection(I2S_Type *base, sai_master_slave_t masterSlave)`

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- `base` – SAI base pointer.
- `masterSlave` – reference `sai_master_slave_t`.

`static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t masterSlave)`

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- `base` – SAI base pointer.
- `masterSlave` – reference `sai_master_slave_t`.

`static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, sai_master_slave_t masterSlave)`

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- `base` – SAI base pointer.
- `masterSlave` – reference `sai_master_slave_t`.

```
void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,  
                           uint32_t bitWidth, uint32_t channelNumbers)
```

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

```
void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate,  
                           uint32_t bitWidth, uint32_t channelNumbers)
```

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

```
void SAI_TxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t  
                             *config)
```

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

```
void SAI_RxSetBitclockConfig(I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t  
                             *config)
```

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

```
void SAI_SetMasterClockConfig(I2S_Type *base, sai_master_clock_t *config)
```

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

```
void SAI_TxSetFifoConfig(I2S_Type *base, sai_fifo_t *config)
```

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.

- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

```
void SAI_GetClassicI2SConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                             sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetLeftJustifiedConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                                sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetRightJustifiedConfig(sai_transceiver_t *config, sai_word_width_t bitWidth,  
                                 sai_mono_stereo_t mode, uint32_t saiChannelMask)
```

Get right justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetTDMConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,  
                      sai_word_width_t bitWidth, uint32_t dataWordNum, uint32_t  
                      saiChannelMask)
```

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

```
void SAI_GetDSPConfig(sai_transceiver_t *config, sai_frame_sync_len_t frameSyncWidth,  
                      sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t  
                      saiChannelMask)
```

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

```
static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

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

```
static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
 - kSAI_WordStartFlag
 - kSAI_SyncErrorFlag
 - kSAI_FIFOErrorFlag

```
static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)
```

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

```
static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)
```

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer

- **mask** – State mask. It can be a combination of the following sources if defined.
 - `kSAI_WordStartFlag`
 - `kSAI_SyncErrorFlag`
 - `kSAI_FIFOErrorFlag`

`void SAI_TxSoftwareReset(I2S_Type *base, sai_reset_type_t resetType)`

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- **base** – SAI base pointer
- **resetType** – Reset type, FIFO reset or software reset

`void SAI_RxSoftwareReset(I2S_Type *base, sai_reset_type_t resetType)`

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- **base** – SAI base pointer
- **resetType** – Reset type, FIFO reset or software reset

`void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)`

Set the Tx channel FIFO enable mask.

Parameters

- **base** – SAI base pointer
- **mask** – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

`void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)`

Set the Rx channel FIFO enable mask.

Parameters

- **base** – SAI base pointer
- **mask** – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

`void SAI_TxSetDataOrder(I2S_Type *base, sai_data_order_t order)`

Set the Tx data order.

Parameters

- **base** – SAI base pointer
- **order** – Data order MSB or LSB

`void SAI_RxSetDataOrder(I2S_Type *base, sai_data_order_t order)`

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai_fifo_packing_t*.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of *sai_fifo_packing_t*.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
 - kSAI_WordStartInterruptEnable
 - kSAI_SyncErrorInterruptEnable
 - kSAI_FIFOWarningInterruptEnable
 - kSAI_FIFORequestInterruptEnable
 - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
 - kSAI_WordStartInterruptEnable
 - kSAI_SyncErrorInterruptEnable
 - kSAI_FIFOWarningInterruptEnable
 - kSAI_FIFORequestInterruptEnable
 - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
 - kSAI_WordStartInterruptEnable
 - kSAI_SyncErrorInterruptEnable
 - kSAI_FIFOWarningInterruptEnable
 - kSAI_FIFORequestInterruptEnable
 - kSAI_FIFOErrorInterruptEnable


```
static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)
```

Disables the SAI Rx interrupt requests.

Parameters

- `base` – SAI base pointer
- `mask` – interrupt source The parameter can be a combination of the following sources if defined.
 - `kSAI_WordStartInterruptEnable`
 - `kSAI_SyncErrorInterruptEnable`
 - `kSAI_FIFOWarningInterruptEnable`
 - `kSAI_FIFORequestInterruptEnable`
 - `kSAI_FIFOErrorInterruptEnable`

```
static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)
```

Enables/disables the SAI Tx DMA requests.

Parameters

- `base` – SAI base pointer
- `mask` – DMA source The parameter can be combination of the following sources if defined.
 - `kSAI_FIFOWarningDMAEnable`
 - `kSAI_FIFORequestDMAEnable`
- `enable` – True means enable DMA, false means disable DMA.

```
static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)
```

Enables/disables the SAI Rx DMA requests.

Parameters

- `base` – SAI base pointer
- `mask` – DMA source The parameter can be a combination of the following sources if defined.
 - `kSAI_FIFOWarningDMAEnable`
 - `kSAI_FIFORequestDMAEnable`
- `enable` – True means enable DMA, false means disable DMA.

```
static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)
```

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- `base` – SAI base pointer.
- `channel` – Which data channel used.

Returns

data register address.

```
static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)
```

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

```
void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer,  
                      uint32_t size)
```

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

```
void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask,  
                                  uint32_t bitWidth, uint8_t *buffer, uint32_t size)
```

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

```
static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)
```

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

```
void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer,  
                     uint32_t size)
```

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask,  
                                uint32_t bitWidth, uint8_t *buffer, uint32_t size)
```

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

```
static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)
```

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

```
void SAI_TransferTxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
                              callback, void *userData)
```

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

```
void SAI_TransferRxCreateHandle(I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t  
                              callback, void *userData)
```

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.

- `handle` – SAI handle pointer.
- `callback` – Pointer to the user callback function.
- `userData` – User parameter passed to the callback function.

`void SAI_TransferTxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)`
SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI handle pointer.
- `config` – transmitter configurations.

`void SAI_TransferRxSetConfig(I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)`
SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI handle pointer.
- `config` – receiver configurations.

`status_t SAI_TransferSendNonBlocking(I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)`

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the `SAI_TxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

Parameters

- `base` – SAI base pointer.
- `handle` – Pointer to the `sai_handle_t` structure which stores the transfer state.
- `xfer` – Pointer to the `sai_transfer_t` structure.

Return values

- `kStatus_Success` – Successfully started the data receive.
- `kStatus_SAI_TxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)`

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the `SAI_RxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the `sai_handle_t` structure which stores the transfer state.
- xfer – Pointer to the `sai_transfer_t` structure.

Return values

- `kStatus_Success` – Successfully started the data receive.
- `kStatus_SAI_RxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`status_t` SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, `size_t` *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the `sai_handle_t` structure which stores the transfer state.
- count – Bytes count sent.

Return values

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

`status_t` SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, `size_t` *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the `sai_handle_t` structure which stores the transfer state.
- count – Bytes count received.

Return values

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

`void` SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer.
- handle – Pointer to the `sai_handle_t` structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer
- handle – Pointer to the *sai_handle_t* structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the *sai_handle_t* structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the *sai_handle_t* structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.10

`_sai_status_t`, SAI return status.

Values:

enumerator `kStatus_SAI_TxBusy`

SAI Tx is busy.

enumerator `kStatus_SAI_RxBusy`

SAI Rx is busy.

enumerator `kStatus_SAI_TxError`

SAI Tx FIFO error.

enumerator `kStatus_SAI_RxError`

SAI Rx FIFO error.

enumerator `kStatus_SAI_QueueFull`

SAI transfer queue is full.

enumerator `kStatus_SAI_TxIdle`

SAI Tx is idle

enumerator `kStatus_SAI_RxIdle`

SAI Rx is idle

`_sai_channel_mask`, sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator `kSAI_Channel0Mask`

channel 0 mask value

enumerator `kSAI_Channel1Mask`

channel 1 mask value

enumerator `kSAI_Channel2Mask`

channel 2 mask value

enumerator `kSAI_Channel3Mask`

channel 3 mask value

enumerator `kSAI_Channel4Mask`

channel 4 mask value

enumerator `kSAI_Channel5Mask`

channel 5 mask value

enumerator `kSAI_Channel6Mask`

channel 6 mask value

enumerator `kSAI_Channel7Mask`

channel 7 mask value

enum `_sai_protocol`

Define the SAI bus type.

Values:

enumerator `kSAI_BusLeftJustified`

Uses left justified format.

enumerator kSAI_BusRightJustified

Uses right justified format.

enumerator kSAI_BusI2S

Uses I2S format.

enumerator kSAI_BusPCMA

Uses I2S PCM A format.

enumerator kSAI_BusPCMB

Uses I2S PCM B format.

enum _sai_master_slave

Master or slave mode.

Values:

enumerator kSAI_Master

Master mode include bclk and frame sync

enumerator kSAI_Slave

Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave

bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master

bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo

Mono or stereo audio format.

Values:

enumerator kSAI_Stereo

Stereo sound.

enumerator kSAI_MonoRight

Only Right channel have sound.

enumerator kSAI_MonoLeft

Only left channel have sound.

enum _sai_data_order

SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB

LSB bit transferred first

enumerator kSAI_DataMSB

MSB bit transferred first

enum _sai_clock_polarity

SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh

Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow

Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge

Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge

Sample inputs on rising edge

enum _sai_sync_mode

Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync

Asynchronous mode

enumerator kSAI_ModeSync

Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx

Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx

Synchronous with another SAI receiver

enum _sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1

Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable

FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable
FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type

The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing

The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate

Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz

Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz

Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz

Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz

Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz

Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz

Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz

Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz

Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz

Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz

Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz

Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz

Sample rate 384000 Hz

enum _sai_word_width

Audio word width.

Values:

enumerator kSAI_WordWidth8bits

Audio data width 8 bits

enumerator kSAI_WordWidth16bits

Audio data width 16 bits

enumerator kSAI_WordWidth24bits

Audio data width 24 bits

enumerator kSAI_WordWidth32bits

Audio data width 32 bits

enum _sai_data_pin_state

sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState

transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero

transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine

sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled

sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead

sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite

sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite

sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead

sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite

sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type

sai transceiver type

Values:

enumerator kSAI_Transmitter

sai transmitter

enumerator kSAI_Receiver

sai receiver

enum _sai_frame_sync_len

sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk

1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth

Frame sync length decided by word width

typedef enum _sai_protocol sai_protocol_t

Define the SAI bus type.

typedef enum _sai_master_slave sai_master_slave_t

Master or slave mode.

typedef enum _sai_mono_stereo sai_mono_stereo_t

Mono or stereo audio format.

typedef enum _sai_data_order sai_data_order_t

SAI data order, MSB or LSB.

`typedef enum _sai_clock_polarity sai_clock_polarity_t`
SAI clock polarity, active high or low.

`typedef enum _sai_sync_mode sai_sync_mode_t`
Synchronous or asynchronous mode.

`typedef enum _sai_bclk_source sai_bclk_source_t`
Bit clock source.

`typedef enum _sai_reset_type sai_reset_type_t`
The reset type.

`typedef enum _sai_fifo_packing sai_fifo_packing_t`
The SAI packing mode The mode includes 8 bit and 16 bit packing.

`typedef struct _sai_config sai_config_t`
SAI user configuration structure.

`typedef enum _sai_sample_rate sai_sample_rate_t`
Audio sample rate.

`typedef enum _sai_word_width sai_word_width_t`
Audio word width.

`typedef enum _sai_data_pin_state sai_data_pin_state_t`
sai data pin state definition

`typedef enum _sai_fifo_combine sai_fifo_combine_t`
sai fifo combine mode definition

`typedef enum _sai_transceiver_type sai_transceiver_type_t`
sai transceiver type

`typedef enum _sai_frame_sync_len sai_frame_sync_len_t`
sai frame sync len

`typedef struct _sai_transfer_format sai_transfer_format_t`
sai transfer format

`typedef struct _sai_master_clock sai_master_clock_t`
master clock configurations

`typedef struct _sai_fifo sai_fifo_t`
sai fifo configurations

`typedef struct _sai_bit_clock sai_bit_clock_t`
sai bit clock configurations

`typedef struct _sai_frame_sync sai_frame_sync_t`
sai frame sync configurations

`typedef struct _sai_serial_data sai_serial_data_t`
sai serial data configurations

`typedef struct _sai_transceiver sai_transceiver_t`
sai transceiver configurations

`typedef struct _sai_transfer sai_transfer_t`
SAI transfer structure.

`typedef struct _sai_handle sai_handle_t`

```
typedef void (*sai_transfer_callback_t)(I2S_Type *base, sai_handle_t *handle, status_t status, void *userData)
```

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

struct __sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol

Audio bus protocol in SAI

sai_sync_mode_t syncMode

SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource

Bit Clock source

sai_master_slave_t masterSlave

Master or slave

struct __sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz

Sample rate of audio data

uint32_t bitWidth

Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo

Mono or stereo

`uint32_t` masterClockHz
Master clock frequency in Hz

`uint8_t` watermark
Watermark value

`uint8_t` channel
Transfer start channel

`uint8_t` channelMask
enabled channel mask value, reference `_sai_channel_mask`

`uint8_t` endChannel
end channel number

`uint8_t` channelNums
Total enabled channel numbers

`sai_protocol_t` protocol
Which audio protocol used

`bool` isFrameSyncCompact
True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

`struct _sai_master_clock`
#include <fsl_sai.h> master clock configurations

Public Members

`bool` mclkOutputEnable
master clock output enable

`uint32_t` mclkHz
target mclk frequency

`uint32_t` mclkSourceClkHz
mclk source frequency

`struct _sai_fifo`
#include <fsl_sai.h> sai fifo configurations

Public Members

`bool` fifoContinueOnError
fifo continues when error occur

`sai_fifo_combine_t` fifoCombine
fifo combine mode

`sai_fifo_packing_t` fifoPacking
fifo packing mode

`uint8_t` fifoWatermark
fifo watermark

`struct _sai_bit_clock`
#include <fsl_sai.h> sai bit clock configurations

Public Members

bool bclkInputDelay

bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time

.

sai_clock_polarity_t bclkPolarity

bit clock polarity

sai_bclk_source_t bclkSource

bit Clock source

struct *_sai_frame_sync*

#include <fsl_sai.h> sai frame sync configurations

Public Members

uint8_t frameSyncWidth

frame sync width in number of bit clocks

bool frameSyncEarly

TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand

internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity

frame sync polarity

struct *_sai_serial_data*

#include <fsl_sai.h> sai serial data configurations

Public Members

sai_data_pin_state_t dataMode

sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder

configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length

configure the number of bits in the first word in each frame

uint8_t dataWordNLength

configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength

used to record the data length for dma transfer

uint8_t dataFirstBitShifted

Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum

configure the number of words in each frame

uint32_t dataMaskedWord

configure whether the transmit word is masked

struct *_sai_transceiver*

#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference `_sai_channel_mask`

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct `_sai_transfer`
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct `_sai_handle`
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
Transfer status

sai_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

`uint8_t bitWidth`
Bit width for transfer, 8/16/24/32 bits

`uint8_t channel`
Transfer start channel

`uint8_t channelMask`
enabled channel mask value, refernece `_sai_channel_mask`

`uint8_t endChannel`
end channel number

`uint8_t channelNums`
Total enabled channel numbers

`sai_transfer_t saiQueue[(4U)]`
Transfer queue storing queued transfer

`size_t transferSize[(4U)]`
Data bytes need to transfer

`volatile uint8_t queueUser`
Index for user to queue transfer

`volatile uint8_t queueDriver`
Index for driver to get the transfer data and size

`uint8_t watermark`
Watermark value

2.79 SAI EDMA Driver

```
void SAI_TransferTxCreateHandleEDMA(I2S_Type *base, sai_edma_handle_t *handle,  
                                   sai_edma_callback_t callback, void *userData,  
                                   edma_handle_t *txDmaHandle)
```

Initializes the SAI eDMA handle.

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

Parameters

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.
- `callback` – Pointer to user callback function.
- `userData` – User parameter passed to the callback function.
- `txDmaHandle` – eDMA handle pointer; this handle shall be static allocated by users.

```
void SAI_TransferRxCreateHandleEDMA(I2S_Type *base, sai_edma_handle_t *handle,  
                                   sai_edma_callback_t callback, void *userData,  
                                   edma_handle_t *rxDmaHandle)
```

Initializes the SAI Rx eDMA handle.

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

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- rxDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

```
void SAI_TransferSetInterleaveType(sai_edma_handle_t *handle, sai_edma_interleave_t
                                interleaveType)
```

Initializes the SAI interleave type.

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI_EDMAInterleavePerChannelSample always

Parameters

- handle – SAI eDMA handle pointer.
- interleaveType – Multi channel interleave type.

```
void SAI_TransferTxSetConfigEDMA(I2S_Type *base, sai_edma_handle_t *handle,
                                sai_transceiver_t *saiConfig)
```

Configures the SAI Tx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of *_sai_channel_mask* enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnWrite to the fifoCombine member of *sai_fifo_combine_t* which is a member of *sai_transceiver_t*. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

```
void SAI_TransferRxSetConfigEDMA(I2S_Type *base, sai_edma_handle_t *handle,
                                sai_transceiver_t *saiConfig)
```

Configures the SAI Rx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of *_sai_channel_mask* enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of *sai_fifo_combine_t*

which is a member of `sai_transceiver_t`. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;  
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_  
↪Channel1Mask);  
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;  
SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

status_t SAI_TransferSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

This function support multi channel transfer,

- a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
- b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

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

status_t SAI_TransferReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using eDMA.

This function support multi channel transfer,

- a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
- b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call the `SAI_GetReceiveRemainingBytes` to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- `base` – SAI base pointer
- `handle` – SAI eDMA handle pointer.
- `xfer` – Pointer to DMA transfer structure.

Return values

- `kStatus_Success` – Start a SAI eDMA receive successfully.
- `kStatus_InvalidArgument` – The input argument is invalid.
- `kStatus_RxBusy` – SAI is busy receiving data.

status_t `SAI_TransferSendLoopEDMA(I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)`

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function `SAI_TransferAbortSendEDMA` to stop the loop transfer.

Note: This function support loop transfer only, such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in `sai_edma_handle_t`, so application could redefine the `SAI_XFER_QUEUE_SIZE` to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.
- `xfer` – Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (`loopTransferCount`).
- `loopTransferCount` – the counts of `xfer` array.

Return values

- `kStatus_Success` – Start a SAI eDMA send successfully.
- `kStatus_InvalidArgument` – The input argument is invalid.

status_t `SAI_TransferReceiveLoopEDMA(I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)`

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function `SAI_TransferAbortReceiveEDMA` to stop the loop transfer.

Note: This function support loop transfer only, such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required,

as the function use the tcd pool in `sai_edma_handle_t`, so application could redefine the `SAI_XFER_QUEUE_SIZE` to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.
- `xfer` – Pointer to the DMA transfer structure, should be a array with elements counts ≥ 1 (`loopTransferCount`).
- `loopTransferCount` – the counts of `xfer` array.

Return values

- `kStatus_Success` – Start a SAI eDMA receive successfully.
- `kStatus_InvalidArgument` – The input argument is invalid.

`void SAI_TransferTerminateSendEDMA(I2S_Type *base, sai_edma_handle_t *handle)`

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call `SAI_TransferAbortSendEDMA`.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.

`void SAI_TransferTerminateReceiveEDMA(I2S_Type *base, sai_edma_handle_t *handle)`

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call `SAI_TransferAbortReceiveEDMA`.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.

`void SAI_TransferAbortSendEDMA(I2S_Type *base, sai_edma_handle_t *handle)`

Aborts a SAI transfer using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call `SAI_TransferTerminateSendEDMA`.

Parameters

- `base` – SAI base pointer.
- `handle` – SAI eDMA handle pointer.

`void SAI_TransferAbortReceiveEDMA(I2S_Type *base, sai_edma_handle_t *handle)`

Aborts a SAI receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call `SAI_TransferTerminateReceiveEDMA`.

Parameters

- `base` – SAI base pointer
- `handle` – SAI eDMA handle pointer.

```
status_t SAI_TransferGetSendCountEDMA(I2S_Type *base, sai_edma_handle_t *handle, size_t
                                     *count)
```

Gets byte count sent by SAI.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- count – Bytes count sent by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

```
status_t SAI_TransferGetReceiveCountEDMA(I2S_Type *base, sai_edma_handle_t *handle, size_t
                                         *count)
```

Gets byte count received by SAI.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- count – Bytes count received by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

```
uint32_t SAI_TransferGetValidTransferSlotsEDMA(I2S_Type *base, sai_edma_handle_t *handle)
```

Gets valid transfer slot.

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

Return values

valid – slot count that application submit.

```
FSL_SAI_EDMA_DRIVER_VERSION
```

Version 2.7.3

```
enum _sai_edma_interleave
```

sai interleave type

Values:

```
enumerator kSAI_EDMAInterleavePerChannelSample
```

```
enumerator kSAI_EDMAInterleavePerChannelBlock
```

```
typedef struct sai_edma_handle sai_edma_handle_t
```

```
typedef void (*sai_edma_callback_t)(I2S_Type *base, sai_edma_handle_t *handle, status_t status, void *userData)
```

SAI eDMA transfer callback function for finish and error.

```
typedef enum _sai_edma_interleave sai_edma_interleave_t  
sai_interleave_type
```

MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferReceiveEDMA

MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferSendEDMA

```
struct sai_edma_handle
```

#include <fsl_sai_edma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for SAI send

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t bytesPerFrame

Bytes in a frame

uint8_t channelMask

Enabled channel mask value, reference *_sai_channel_mask*

uint8_t channelNums

total enabled channel nums

uint8_t channel

Which data channel

uint8_t count

The transfer data count in a DMA request

uint32_t state

Internal state for SAI eDMA transfer

sai_edma_callback_t callback

Callback for users while transfer finish or error occurs

void *userData

User callback parameter

uint8_t tcd[((4U) + 1U) * sizeof(*edma_tcd_t*)]

TCD pool for eDMA transfer.

sai_transfer_t saiQueue[(4U)]

Transfer queue storing queued transfer.

size_t transferSize[(4U)]

Data bytes need to transfer

sai_edma_interleave_t interleaveType

Transfer interleave type

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.80 SMARTDMA: SMART DMA Driver

typedef void (*smartdma_callback_t)(void *param)

Callback function prototype for the smartdma driver.

void SMARTDMA_Init(uint32_t apiMemAddr, const void *firmware, uint32_t
firmwareSizeByte)

Initialize the SMARTDMA.

Deprecated:

Do not use this function. It has been superseded by SMARTDMA_InitWithoutFirmware and SMARTDMA_InstallFirmware.

Parameters

- apiMemAddr – The address firmware will be copied to.
- firmware – The firmware to use.
- firmwareSizeByte – Size of firmware.

void SMARTDMA_InitWithoutFirmware(void)

Initialize the SMARTDMA.

This function is similar with SMARTDMA_Init, the difference is this function does not install the firmware, the firmware could be installed using SMARTDMA_InstallFirmware.

void SMARTDMA_InstallFirmware(uint32_t apiMemAddr, const void *firmware, uint32_t
firmwareSizeByte)

Install the firmware.

Note: Only call this function when SMARTDMA is not busy.

Parameters

- apiMemAddr – The address firmware will be copied to.
- firmware – The firmware to use.
- firmwareSizeByte – Size of firmware.

void SMARTDMA_InstallCallback(smartdma_callback_t callback, void *param)

Install the complete callback function.

Note: Only call this function when SMARTDMA is not busy.

Parameters

- callback – The callback called when smartdma program finished.
- param – Parameter for the callback.

void SMARTDMA_Boot(uint32_t apiIndex, void *pParam, uint8_t mask)

Boot the SMARTDMA to run program.

Note: Only call this function when SMARTDMA is not busy.

Note: The memory *pParam shall not be freed before the SMARTDMA function finished.

Parameters

- apiIndex – Index of the API to call.
- pParam – Pointer to the parameter allocated by caller.
- mask – Value set to register SMARTDMA->ARM2EZH[0:1].

void SMARTDMA_Boot1(uint32_t apiIndex, const smartdma_param_t *pParam, uint8_t mask)

Copy SMARTDMA params and Boot to run program.

This function is similar with SMARTDMA_Boot, the only difference is, this function copies the *pParam to a local variable, upper layer can free the pParam's memory before the SMARTDMA execution finished, for example, upper layer can define the param as a local variable.

Note: Only call this function when SMARTDMA is not busy.

Parameters

- apiIndex – Index of the API to call.
- pParam – Pointer to the parameter.
- mask – Value set to SMARTDMA_ARM2SMARTDMA[0:1].

void SMARTDMA_Deinit(void)

Deinitialize the SMARTDMA.

void SMARTDMA_Reset(void)

Reset the SMARTDMA.

void SMARTDMA_HandleIRQ(void)

SMARTDMA IRQ.

void SMARTDMA_SetExternalFlag(uint8_t flag)

SMARTDMA set EX flag.

void SMARTDMA_AccessShareRAM(uint8_t flag)

SMARTDMA access RAM.

FSL_SMARTDMA_DRIVER_VERSION

SMARTDMA driver version.

2.81 SYSPM: System Performance Monitor

enum __syspm_monitor

syspm select control monitor

Values:

```
    enumerator kSYSPM_Monitor0
        Monitor 0
enum __syspm_event
    syspm select event
    Values:
    enumerator kSYSPM_Event1
        Event 1
    enumerator kSYSPM_Event2
        Event 2
    enumerator kSYSPM_Event3
        Event 3
enum __syspm_mode
    syspm set count mode
    Values:
    enumerator kSYSPM_BothMode
        count in both modes
    enumerator kSYSPM_UserMode
        count only in user mode
    enumerator kSYSPM_PrivilegedMode
        count only in privileged mode
enum __syspm_startstop_control
    syspm start/stop control
    Values:
    enumerator kSYSPM_Idle
        idle >
    enumerator kSYSPM_LocalStop
        local stop
    enumerator kSYSPM_LocalStart
        local start
    enumerator KSYSPM_EnableTraceControl
        enable global TSTART/TSTOP
    enumerator kSYSPM_GlobalStart
        global stop
    enumerator kSYSPM_GlobalStop
        global start
typedef enum __syspm_monitor syspm_monitor_t
    syspm select control monitor
typedef enum __syspm_event syspm_event_t
    syspm select event
typedef enum __syspm_mode syspm_mode_t
    syspm set count mode
```

```
typedef enum _syspm_startstop_control syspm_startstop_control_t  
    syspm_start/stop control
```

```
void SYSPM_Init(SYSPM_Type *base)
```

Initializes the SYSPM.

This function enables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

```
void SYSPM_Deinit(SYSPM_Type *base)
```

Deinitializes the SYSPM.

This function disables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

```
void SYSPM_SelectEvent(SYSPM_Type *base, syspm_monitor_t monitor, syspm_event_t event,  
    uint8_t eventCode)
```

Select event counters.

Parameters

- base – SYSPM peripheral base address.
- event – syspm select event, see to *syspm_event_t*.
- eventCode – select which event to be counted in PMECTR_x., see to table Events.

```
void SYSPM_ResetEvent(SYSPM_Type *base, syspm_monitor_t monitor, syspm_event_t event)
```

Reset event counters.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

```
void SYSPM_ResetInstructionEvent(SYSPM_Type *base, syspm_monitor_t monitor)
```

Reset Instruction Counter.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

```
void SYSPM_SetCountMode(SYSPM_Type *base, syspm_monitor_t monitor, syspm_mode_t  
    mode)
```

Set count mode.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- mode – syspm select counter mode, see to *syspm_mode_t*.

```
void SYSPM_SetStartStopControl(SYSPM_Type *base, syspm_monitor_t monitor,  
    syspm_startstop_control_t ssc)
```

Set Start/Stop Control.

Parameters

- `base` – SYSPM peripheral base address.
- `monitor` – syspm control monitor, see to `syspm_monitor_t`.
- `ssc` – This 3-bit field provides a three-phase mechanism to start/stop the counters. It includes a prioritized scheme with local start > local stop > global start > global stop > conditional TSTART > TSTOP. The global and conditional start/stop affect all configured PM/PSAM module concurrently so counters are “coherent”. see to `syspm_startstop_control_t`

`void SYSPM_DisableCounter(SYSPM_Type *base, syspm_monitor_t monitor)`

Disable Counters if Stopped or Halted.

Parameters

- `base` – SYSPM peripheral base address.
- `monitor` – syspm control monitor, see to `syspm_monitor_t`.

`uint64_t SYSPM_GetEventCounter(SYSPM_Type *base, syspm_monitor_t monitor, syspm_event_t event)`

This is the the 40-bits of eventx counter. The value in this register increments each time the event selected in `PMCRx[SELEVTx]` occurs.

Parameters

- `base` – SYSPM peripheral base address.
- `monitor` – syspm control monitor, see to `syspm_monitor_t`.
- `event` – syspm select event, see to `syspm_event_t`.

Returns

- When the return value is not equal to `SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE`, the return value represents a 40 bits eventx counter.
- When the return value is equal to `SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE`, the return value represents timeout occurred.

`EVENT_COUNT_STABLE_TIMEOUT`

Max loops to wait for SYSPM event count stable (0 means wait forever)

`INSTRUCTION_COUNT_STABLE_TIMEOUT`

Max loops to wait for SYSPM instruction count stable (0 means wait forever)

`FSL_SYSPM_DRIVER_VERSION`

SYSPM driver version.

`SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE`

2.82 Digital Tamper

2.83 TDET

`status_t TDET_Init(DIGTMP_Type *base)`

Initialize TDET.

This function initializes TDET.

Parameters

- `base` – TDET peripheral base address

Returns

Status of the init operation

void TDET_Deinit(DIGTMP_Type *base)

Deinitialize TDET.

This function disables glitch filters and active tampers This function disables the TDET clock and prescaler in TDET Control Register.

Parameters

- base – TDET peripheral base address

void TDET_GetDefaultConfig(DIGTMP_Type *base, *t det_config_t* *defaultConfig)

Gets default values for the TDET Control Register.

This function fills the given structure with default values for the TDET Control Register. The default values are:

```
defaultConfig->innerClockAndPrescalerEnable = true
defaultConfig->tamperForceSystemResetEnable = false
defaultConfig->updateMode = kTDET_StatusLockWithTamper
defaultConfig->clockSourceActiveTamper0 = kTDET_ClockType1Hz
defaultConfig->clockSourceActiveTamper1 = kTDET_ClockType1Hz
defaultConfig->prescaler = 0
```

Parameters

- base – TDET peripheral base address
- defaultConfig – **[out]** Pointer to structure to be filled with default parameters

status_t TDET_SetConfig(DIGTMP_Type *base, const *t det_config_t* *config)

Writes to the TDET Control Register.

This function writes the given structure to the TDET Control Register.

Parameters

- base – TDET peripheral base address
- config – Pointer to structure with TDET peripheral configuration parameters

Returns

kStatus_Fail when writing to TDET Control Register is not allowed

Returns

kStatus_Success when operation completes successfully

status_t TDET_SoftwareReset(DIGTMP_Type *base)

Software reset.

This function resets all TDET registers. The CR[SWR] itself is not affected; it is reset by VBAT POR only.

Parameters

- base – TDET peripheral base address

Returns

kStatus_Fail when writing to TDET Control Register is not allowed

Returns

kStatus_Success when operation completes successfully

```
status_t TDET_ActiveTamperSetConfig(DIGTMP_Type *base, const tdet_active_tamper_config_t
                                   *activeTamperConfig, uint32_t
                                   activeTamperRegisterSelect)
```

Writes to the active tamper register(s).

This function writes per active tamper register parameters to active tamper register(s).

Parameters

- base – TDET peripheral base address
- activeTamperConfig – Pointer to structure with active tamper register parameters
- activeTamperRegisterSelect – Bit mask for active tamper registers to be configured. The passed value is combination of tdet_active_tamper_register_t values (OR'ed).

Returns

kStatus_Fail when writing to TDET Active Tamper Register(s) is not allowed

Returns

kStatus_Success when operation completes successfully

```
void TDET_PinGetDefaultConfig(DIGTMP_Type *base, tdet_pin_config_t *pinConfig)
```

Gets default values for tamper pin configuration.

This function fills the give structure with default values for the tamper pin and glitch filter configuration. The default values are: code pinConfig->pinPolarity = kTDET_TamperPinPolarityExpectNormal; pinConfig->pinDirection = kTDET_TamperPinDirectionIn; pinConfig->tamperPullEnable = false; pinConfig->tamperPinSampleFrequency = kTDET_GlitchFilterSamplingEveryCycle8; pinConfig->tamperPinSampleWidth = kTDET_GlitchFilterSampleDisable; pinConfig->glitchFilterEnable = false; pinConfig->glitchFilterPrescaler = kTDET_GlitchFilterClock512Hz; pinConfig->glitchFilterWidth = 0; pinConfig->tamperPinExpected = kTDET_GlitchFilterExpectedLogicZero; pinConfig->tamperPullSelect = kTDET_GlitchFilterPullTypeAssert; endcode

Parameters

- base – TDET peripheral base address
- pinConfig – **[out]** Pointer to structure to be filled with tamper pins default parameters

```
status_t TDET_PinSetConfig(DIGTMP_Type *base, const tdet_pin_config_t *pinConfig, uint32_t
                           pinSelect)
```

Writes the tamper pin configuration.

This function writes per pin parameters to tamper pin and glitch filter configuration registers.

Parameters

- base – TDET peripheral base address
- pinConfig – Pointer to structure with tamper pin and glitch filter configuration parameters
- pinSelect – Bit mask for tamper pins to be configured. The passed value is combination of enum_tdet_tamper_pin (tdet_tamper_pin_t) values (OR'ed).

Returns

kStatus_Fail when writing to TDET Pin Direction, Pin Polarity or Glitch Filter Register(s) is not allowed

Returns

kStatus_Success when operation completes successfully

status_t TDET_GetStatusFlags(DIGTMP_Type *base, uint32_t *result)

Reads the Status Register.

This function reads flag bits from TDET Status Register.

Parameters

- base – TDET peripheral base address
- result – **[out]** Pointer to uint32_t where to write Status Register read value. Use tdet_status_flag_t to decode individual flags.

Returns

kStatus_Fail when Status Register reading is not allowed

Returns

kStatus_Success when result is written with the Status Register read value

status_t TDET_ClearStatusFlags(DIGTMP_Type *base, uint32_t mask)

Writes to the Status Register.

This function clears specified flag bits in TDET Status Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the flag bits to be cleared. Use tdet_status_flag_t to encode flags.

Returns

kStatus_Fail when Status Register writing is not allowed

Returns

kStatus_Success when mask is written to the Status Register

status_t TDET_EnableInterrupts(DIGTMP_Type *base, uint32_t mask)

Writes to the Interrupt Enable Register.

This function sets specified interrupt enable bits in TDET Interrupt Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the interrupt enable bits to be set.

Returns

kStatus_Fail when Interrupt Enable Register writing is not allowed

Returns

kStatus_Success when mask is written to the Interrupt Enable Register

status_t TDET_DisableInterrupts(DIGTMP_Type *base, uint32_t mask)

Writes to the Interrupt Enable Register.

This function clears specified interrupt enable bits in TDET Interrupt Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the interrupt enable bits to be cleared.

Returns

kStatus_Fail when Interrupt Enable Register writing is not allowed

Returns

kStatus_Success when specified bits are cleared in the Interrupt Enable Register

status_t TDET__EnableTamper(DIGTMP_Type *base, uint32_t mask)

Writes to the Tamper Enable Register.

This function sets specified tamper enable bits in TDET Tamper Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the tamper enable bits to be set.

Returns

kStatus_Fail when Tamper Enable Register writing is not allowed

Returns

kStatus_Success when mask is written to the Tamper Enable Register

status_t TDET__DisableTamper(DIGTMP_Type *base, uint32_t mask)

Writes to the Tamper Enable Register.

This function clears specified tamper enable bits in TDET Tamper Enable Register.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the tamper enable bits to be cleared.

Returns

kStatus_Fail when Tamper Enable Register writing is not allowed

Returns

kStatus_Success when specified bits are cleared in the Tamper Enable Register

status_t TDET__ForceTamper(DIGTMP_Type *base)

Writes to the Tamper Seconds Register.

This function writes to TDET Tamper Seconds Register. This causes Status Register DTF flag to be set (TDET tampering detected).

Parameters

- base – TDET peripheral base address

Returns

kStatus_Fail when Tamper Seconds Register writing is not allowed

Returns

kStatus_Success when Tamper Seconds Register is written

status_t TDET__GetTamperTimeSeconds(DIGTMP_Type *base, uint32_t *tamperTimeSeconds)

Reads the Tamper Seconds Register.

This function reads TDET Tamper Seconds Register. The read value returns the time in seconds at which the Status Register DTF flag was set.

Parameters

- base – TDET peripheral base address
- tamperTimeSeconds – Time in seconds at which the tamper detection SR[DTF] flag was set.

Returns

kStatus_Fail when Tamper Seconds Register reading is not allowed

Returns

kStatus_Success when Tamper Seconds Register is read

void TDET_LockRegisters(DIGTMP_Type *base, uint32_t mask)

Writes to the TDET Lock Register.

This function clears specified lock bits in the TDET Lock Register. When a lock bit is clear, a write to corresponding TDET Register is ignored. Once cleared, these bits can only be set by VBAT POR or software reset.

Parameters

- base – TDET peripheral base address
- mask – Bit mask for the lock bits to be cleared. Use tdet_register_t values to encode (OR'ed) which TDET Registers shall be locked.

FSL_TDET_DRIVER_VERSION

Defines TDET driver version 2.3.0.

Change log:

- Version 2.3.0
 - Added enum for TIF10.
- Version 2.2.0
 - Added support for chips without active tamper pins.
- Version 2.1.1
 - Added clearing SR_TAF and SR_DTF into TDET_Init().
 - Fix typo in kTDET_ClockType64Hz comment
- Version 2.1.0
 - Added setting of disabling prescaler on tamper event into TDET_SetConfig() and TDET_GetDefaultConfig functions.
- Version 2.0.0
 - Initial version

enum __tdet_update_mode

TDET Update Mode.

These constants allow TDET interrupts to be cleared if no tampering has been detected, while still preventing the TDET Tamper Flag (SR[DTF]) from being cleared once it is set.

Values:

enumerator kTDET_StatusLockNormal

TDET Status Register cannot be written when the Status Register Lock bit within the Lock Register (LR[SRL]) is clear

enumerator kTDET_StatusLockWithTamper

TDET Status Register cannot be written when the Status Register Lock bit within the Lock Register (LR[SRL]) is clear and TDET Tamper Flag (SR[DTF]) is set

enum __tdet_active_tamper_clock

TDET Active Tamper Clock Source.

These constants define the clock source for Active Tamper Shift Register to configure in a TDET base.

Values:

enumerator kTDET_ClockType1Hz

clocked by 1 Hz prescaler clock

enumerator kTDET_ClockType64Hz

clocked by 64 Hz prescaler clock

enum __tdet_pin_polarity

TDET Tamper Pin Polarity.

These constants define tamper pin polarity to configure in a TDET base.

Values:

enumerator kTDET_TamperPinPolarityExpectNormal

Tamper pin expected value is not inverted

enumerator kTDET_TamperPinPolarityExpectInverted

Tamper pin expected value is inverted

enum __tdet_pin_direction

TDET Tamper Pin Direction.

These constants define tamper pin direction to configure in a TDET base.

Values:

enumerator kTDET_TamperPinDirectionIn

Tamper pins configured as input

enumerator kTDET_TamperPinDirectionOut

Tamper pins configured as output, drives inverse of expected value

enum __tdet_glitch_filter_sample_freq

TDET Glitch Filter Tamper Pin Sample Frequency.

These constants define tamper pin glitch filter sample frequency to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterSamplingEveryCycle8

Sample once every 8 cycles

enumerator kTDET_GlitchFilterSamplingEveryCycle32

Sample once every 32 cycles

enumerator kTDET_GlitchFilterSamplingEveryCycle128

Sample once every 128 cycles

enumerator kTDET_GlitchFilterSamplingEveryCycle512

Sample once every 512 cycles

enum __tdet_glitch_filter_sample_width

TDET Glitch Filter Tamper Pin Sample Width.

These constants define tamper pin glitch filter sample width to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterSampleDisable

Sampling disabled

enumerator kTDET_GlitchFilterSampleCycle2

Sample width pull enable/input buffer enable=2 cycles/1 cycle

enumerator kTDET_GlitchFilterSampleCycle4

Sample width pull enable/input buffer enable=4 cycles/2 cycles

enumerator kTDET_GlitchFilterSampleCycle8

Sample width pull enable/input buffer enable=8 cycles/4 cycles

enum __tdet_glitch_filter_prescaler

TDET Glitch Filter Tamper Pin Clock Source.

These constants define tamper pin glitch filter clock source to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterClock512Hz

Glitch Filter on tamper pin is clocked by the 512 Hz prescaler clock

enumerator kTDET_GlitchFilterClock32768Hz

Glitch Filter on tamper pin is clocked by the 32768 Hz prescaler clock

enum __tdet_glitch_filter_expected

TDET Glitch Filter Tamper Pin Expected Value.

These constants define tamper pin glitch filter expected value to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterExpectedLogicZero

Expected value is logic zero

enumerator kTDET_GlitchFilterExpectedActTamperOut0

Expected value is active tamper 0 output

enumerator kTDET_GlitchFilterExpectedActTamperOut1

Expected value is active tamper 1 output

enumerator kTDET_GlitchFilterExpectedActTamperOutXOR

Expected value is active tamper 0 output XORed with active tamper 1 output

enum __tdet_glitch_filter_pull

TDET Glitch Filter Tamper Pull Select.

These constants define tamper pin glitch filter pull direction to configure in a TDET base.

Values:

enumerator kTDET_GlitchFilterPullTypeAssert

Tamper pin pull direction always asserts the tamper pin.

enumerator kTDET_GlitchFilterPullTypeNegate

Tamper pin pull direction always negates the tamper pin.

enum __tdet_external_tamper_pin

List of TDET external tamperers.

Values:

enumerator kTDET_ExternalTamper0

enumerator kTDET_ExternalTamper1

enumerator kTDET_ExternalTamper2

enumerator kTDET_ExternalTamper3

enumerator kTDET_ExternalTamper4

enumerator kTDET_ExternalTamper5

enumerator kTDET_ExternalTamper6

enumerator kTDET_ExternalTamper7

enum _tdet_active_tamper_register

TDET Active Tamper Register Select.

These constants are used to define activeTamperRegisterSelect argument to be used with TDET_ActiveTamperConfigure().

Values:

enumerator kTDET_ActiveTamperRegister0

enumerator kTDET_ActiveTamperRegister1

enum _tdet_status_flag

TDET Status Register flags.

This provides constants for the TDET Status Register.

Values:

enumerator kTDET_StatusTamperFlag

TDET Digital Tamper Flag

enumerator kTDET_StatusTamperAcknowledgeFlag

TDET Tamper Acknowledge Flag

enumerator kTDET_TIF0

TDET Tamper input 1

enumerator kTDET_TIF1

TDET Tamper input 1

enumerator kTDET_TIF2

TDET Tamper input 2

enumerator kTDET_TIF3

TDET Tamper input 3

enumerator kTDET_TIF4

TDET Tamper input 4

enumerator kTDET_TIF5

TDET Tamper input 5

enumerator kTDET_TIF6

TDET Tamper input 6

enumerator kTDET_TIF7

TDET Tamper input 7

enumerator kTDET_TIF8

TDET Tamper input 8

enumerator kTDET_TIF9

TDET Tamper input 9

enumerator kTDET_StatusTamperPinTamper0

TDET Tamper Pin 0 Tamper detected

```
enumerator kTDET_StatusTamperPinTamper1
    TDET Tamper Pin 1 Tamper detected
enumerator kTDET_StatusTamperPinTamper2
    TDET Tamper Pin 2 Tamper detected
enumerator kTDET_StatusTamperPinTamper3
    TDET Tamper Pin 3 Tamper detected
enumerator kTDET_StatusTamperPinTamper4
    TDET Tamper Pin 4 Tamper detected
enumerator kTDET_StatusTamperPinTamper5
    TDET Tamper Pin 5 Tamper detected
enumerator kTDET_StatusAll
    Mask for all of the TDET Status Register bits
```

```
enum __tdet_interrupt
```

TDET Interrupt Enable Register.

This provides constants for the TDET Interrupt Enable Register.

Values:

```
enumerator kTDET_InterruptTamper
    TDET Digital Tamper Interrupt
enumerator kTDET_InterruptTIF0
    TDET TIF0 Interrupt
enumerator kTDET_InterruptTIF1
    TDET TIF1 Interrupt
enumerator kTDET_InterruptTIF2
    TDET TIF2 Interrupt
enumerator kTDET_InterruptTIF3
    TDET TIF3 Interrupt
enumerator kTDET_InterruptTIF4
    TDET TIF4 Interrupt
enumerator kTDET_InterruptTIF5
    TDET TIF5 Interrupt
enumerator kTDET_InterruptTIF6
    TDET TIF6 Interrupt
enumerator kTDET_InterruptTIF7
    TDET TIF7 Interrupt
enumerator kTDET_InterruptTIF8
    TDET TIF8 Interrupt
enumerator kTDET_InterruptTIF9
    TDET TIF9 Interrupt
enumerator kTDET_InterruptTamperPinTamper0
    TDET Tamper Pin Tamper 0 Interrupt
enumerator kTDET_InterruptTamperPinTamper1
    TDET Tamper Pin Tamper 1 Interrupt
```

enumerator kTDET_InterruptTamperPinTamper2

TDET Tamper Pin Tamper 2 Interrupt

enumerator kTDET_InterruptTamperPinTamper3

TDET Tamper Pin Tamper 3 Interrupt

enumerator kTDET_InterruptTamperPinTamper4

TDET Tamper Pin Tamper 4 Interrupt

enumerator kTDET_InterruptTamperPinTamper5

TDET Tamper Pin Tamper 5 Interrupt

enumerator kTDET_InterruptTamperPinTamper_All

TDET All Tamper Pins Interrupt

enumerator kTDET_InterruptAll

Mask to select all TDET Interrupt Enable Register bits

enum __tdet__tamper

TDET Tamper Enable Register.

This provides constants for the TDET Tamper Enable Register.

Values:

enumerator kTDET_TamperTIF0

TIF0 Tamper Enable

enumerator kTDET_TamperTIF1

TIF1 Tamper Enable

enumerator kTDET_TamperTIF2

TIF2 Tamper Enable

enumerator kTDET_TamperTIF3

TIF3 Tamper Enable

enumerator kTDET_TamperTIF4

TIF4 Tamper Enable

enumerator kTDET_TamperTIF5

TIF5 Tamper Enable

enumerator kTDET_TamperTIF6

TIF6 Tamper Enable

enumerator kTDET_TamperTIF7

TIF7 Tamper Enable

enumerator kTDET_TamperTIF8

TIF8 Tamper Enable

enumerator kTDET_TamperTIF9

TIF9 Tamper Enable

enumerator kTDET_TamperTamperPin0

Tamper Pin 0 Tamper Enable

enumerator kTDET_TamperTamperPin1

Tamper Pin 1 Tamper Enable

enumerator kTDET_TamperTamperPin2

Tamper Pin 2 Tamper Enable

enumerator kTDET_TamperTamperPin3
Tamper Pin 3 Tamper Enable

enumerator kTDET_TamperTamperPin4
Tamper Pin 4 Tamper Enable

enumerator kTDET_TamperTamperPin5
Tamper Pin 5 Tamper Enable

enumerator kTDET_TamperTamperPinAll
All Tamper Pin Tamper Enable

enumerator kTDET_TamperAll
Mask to select all Tamper Enable Register bits

enum _tdet_register

TDET Registers.

This provides constants to encode a mask for the TDET Registers.

Values:

enumerator kTDET_NoRegister
No Register

enumerator kTDET_Control
Control Register

enumerator kTDET_Status
Status Register

enumerator kTDET_Lock
Lock Register

enumerator kTDET_InterruptEnable
Interrupt Enable Register

enumerator kTDET_TamperSeconds
Tamper Seconds Register

enumerator kTDET_TamperEnable
Tamper Enable Register

enumerator kTDET_PinPolarity
Pin Polarity Register

enumerator kTDET_GlitchFilter0
Glitch Filter Register 0

enumerator kTDET_GlitchFilter1
Glitch Filter Register 1

enumerator kTDET_GlitchFilter2
Glitch Filter Register 2

enumerator kTDET_GlitchFilter3
Glitch Filter Register 3

enumerator kTDET_GlitchFilter4
Glitch Filter Register 4

enumerator kTDET_GlitchFilter5
Glitch Filter Register 5

enumerator `kTDET_PinConfigurationRegisters`

Mask to select all TDET Pin Configuration Registers

enumerator `kTDET_AllRegisters`

Mask to select all TDET Registers

typedef enum `_tdet_update_mode` `tdet_update_mode_t`

TDET Update Mode.

These constants allow TDET interrupts to be cleared if no tampering has been detected, while still preventing the TDET Tamper Flag (SR[DTF]) from being cleared once it is set.

typedef enum `_tdet_active_tamper_clock` `tdet_active_tamper_clock_t`

TDET Active Tamper Clock Source.

These constants define the clock source for Active Tamper Shift Register to configure in a TDET base.

typedef struct `_tdet_config` `tdet_config_t`

TDET Control Register.

This structure defines values for TDET Control Register.

typedef enum `_tdet_pin_polarity` `tdet_pin_polarity_t`

TDET Tamper Pin Polarity.

These constants define tamper pin polarity to configure in a TDET base.

typedef enum `_tdet_pin_direction` `tdet_pin_direction_t`

TDET Tamper Pin Direction.

These constants define tamper pin direction to configure in a TDET base.

typedef enum `_tdet_glitch_filter_sample_freq` `tdet_glitch_filter_sample_freq_t`

TDET Glitch Filter Tamper Pin Sample Frequency.

These constants define tamper pin glitch filter sample frequency to configure in a TDET base.

typedef enum `_tdet_glitch_filter_sample_width` `tdet_glitch_filter_sample_width_t`

TDET Glitch Filter Tamper Pin Sample Width.

These constants define tamper pin glitch filter sample width to configure in a TDET base.

typedef enum `_tdet_glitch_filter_prescaler` `tdet_glitch_filter_prescaler_t`

TDET Glitch Filter Tamper Pin Clock Source.

These constants define tamper pin glitch filter clock source to configure in a TDET base.

typedef enum `_tdet_glitch_filter_expected` `tdet_glitch_filter_expected_t`

TDET Glitch Filter Tamper Pin Expected Value.

These constants define tamper pin glitch filter expected value to configure in a TDET base.

typedef enum `_tdet_glitch_filter_pull` `tdet_glitch_filter_pull_t`

TDET Glitch Filter Tamper Pull Select.

These constants define tamper pin glitch filter pull direction to configure in a TDET base.

typedef struct `_tdet_pin_config` `tdet_pin_config_t`

TDET Tamper Pin configuration registers.

This structure defines values for TDET Pin Direction, Pin Polarity, and Glitch Filter registers.

typedef enum `_tdet_external_tamper_pin` `tdet_external_tamper_pin_t`

List of TDET external tampers.

```
typedef enum _tdet_active_tamper_register tdet_active_tamper_register_t
```

TDET Active Tamper Register Select.

These constants are used to define activeTamperRegisterSelect argument to be used with TDET_ActiveTamperConfigure().

```
typedef struct _tdet_active_tamper_config tdet_active_tamper_config_t
```

TDET Active Tamper registers.

This structure defines values for TDET Active Tamper Registers.

```
typedef enum _tdet_status_flag tdet_status_flag_t
```

TDET Status Register flags.

This provides constants for the TDET Status Register.

```
typedef enum _tdet_interrupt tdet_interrupt_t
```

TDET Interrupt Enable Register.

This provides constants for the TDET Interrupt Enable Register.

```
typedef enum _tdet_tamper tdet_tamper_t
```

TDET Tamper Enable Register.

This provides constants for the TDET Tamper Enable Register.

```
typedef enum _tdet_register tdet_register_t
```

TDET Registers.

This provides constants to encode a mask for the TDET Registers.

```
void VBAT0_DriverIRQHandler(void)
```

```
struct _tdet_config
```

```
#include <fsl_tdet.h> TDET Control Register.
```

This structure defines values for TDET Control Register.

Public Members

```
bool innerClockAndPrescalerEnable
```

Enable/disable 32768 Hz clock within TDET and the TDET prescaler that generates 512 Hz, 64Hz and 1 Hz prescaler clocks

```
bool tamperForceSystemResetEnable
```

Enable/disable assertion of chip reset when tampering is detected

```
enum _tdet_update_mode updateMode
```

Selects update mode for TDET Status Register

```
enum _tdet_active_tamper_clock clockSourceActiveTamper0
```

Selects clock source for Active Tamper Shift Register 0

```
enum _tdet_active_tamper_clock clockSourceActiveTamper1
```

Selects clock source for Active Tamper Shift Register 1

```
bool disablePrescalerAfterTamper
```

Allows the 32-KHz clock and prescaler to be automatically disabled after tamper detection and until the system acknowledges the tamper. Disabling the prescaler after detecting a tamper event conserves power and freezes the state of the active tamper outputs and glitch filters. To ensure a clean transition, the prescaler is disabled at the end of a 1 Hz period.

uint32_t prescaler

Initial value for the TDET prescaler 15-bit value.

struct _tdet_pin_config

#include <fsl_tdet.h> TDET Tamper Pin configuration registers.

This structure defines values for TDET Pin Direction, Pin Polarity, and Glitch Filter registers.

Public Members

enum _tdet_pin_polarity pinPolarity

Selects tamper pin expected value

enum _tdet_pin_direction pinDirection

Selects tamper pin direction

bool tamperPullEnable

Enable/disable pull resistor on the tamper pin

enum _tdet_glitch_filter_sample_freq tamperPinSampleFrequency

Selects tamper pin sample frequency

enum _tdet_glitch_filter_sample_width tamperPinSampleWidth

Selects tamper pin sample width

bool glitchFilterEnable

Enable/disable glitch filter on the tamper pin

enum _tdet_glitch_filter_prescaler glitchFilterPrescaler

Selects the prescaler for the glitch filter on tamper pin

uint8_t glitchFilterWidth

6-bit value to configure number of clock edges the input must remain stable for to be passed through the glitch filter for the tamper pin

enum _tdet_glitch_filter_expected tamperPinExpected

Selects tamper pin expected value

enum _tdet_glitch_filter_pull tamperPullSelect

Selects the direction of the tamper pin pull resistor

struct _tdet_active_tamper_config

#include <fsl_tdet.h> TDET Active Tamper registers.

This structure defines values for TDET Active Tamper Registers.

Public Members

uint32_t activeTamperShift

Active tamper shift register. initialize to non-zero value.

uint32_t activeTamperPolynomial

Polynomial of the active tamper shift register.

2.84 TRDC: Trusted Resource Domain Controller

`void TRDC_Init(TRDC_Type *base)`

Initializes the TRDC module.

This function enables the TRDC clock.

Parameters

- `base` – TRDC peripheral base address.

`void TRDC_Deinit(TRDC_Type *base)`

De-initializes the TRDC module.

This function disables the TRDC clock.

Parameters

- `base` – TRDC peripheral base address.

`static inline uint8_t TRDC_GetCurrentMasterDomainId(TRDC_Type *base)`

Gets the domain ID of the current bus master.

Parameters

- `base` – TRDC peripheral base address.

Returns

Domain ID of current bus master.

`void TRDC_GetHardwareConfig(TRDC_Type *base, trdc_hardware_config_t *config)`

Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

- `base` – TRDC peripheral base address.
- `config` – Pointer to the structure to get the configuration.

`static inline void TRDC_SetDacGlobalValid(TRDC_Type *base)`

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- `base` – TRDC peripheral base address.

`static inline void TRDC_LockMasterDomainAssignment(TRDC_Type *base, uint8_t master, uint8_t regNum)`

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

Parameters

- `base` – TRDC peripheral base address.
- `master` – Which master to configure, refer to `trdcx_master_t` in processor header file, `x` is `trdc` instance.
- `regNum` – Which register to configure, processor master can have more than one register for the MDAC configuration.
- `assignIndex` – Which assignment register to lock.

```
static inline void TRDC_SetMasterDomainAssignmentValid(TRDC_Type *base, uint8_t master,
                                                    uint8_t regNum, bool valid)
```

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

Parameters

- `base` – TRDC peripheral base address.
- `master` – Which master to configure.
- `regNum` – Which register to configure, processor master can have more than one register for the MDAC configuration.
- `assignIndex` – Index for the domain assignment register.
- `valid` – True to set valid, false to set invalid.

```
void TRDC_GetDefaultProcessorDomainAssignment(trdc_processor_domain_assignment_t
                                             *domainAssignment)
```

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->domainIdSelect = kTRDC_DidMda;
assignment->lock           = 0U;
```

Parameters

- `domainAssignment` – Pointer to the assignment structure.

```
void TRDC_GetDefaultNonProcessorDomainAssignment(trdc_non_processor_domain_assignment_t
                                                *domainAssignment)
```

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId      = 0U;
assignment->privilegeAttr  = kTRDC_ForceUser;
assignment->secureAttr     = kTRDC_ForceSecure;
assignment->bypassDomainId = 0U;
assignment->lock           = 0U;
```

Parameters

- `domainAssignment` – Pointer to the assignment structure.

```
void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, uint8_t master, uint8_t regNum,
                                       const trdc_processor_domain_assignment_t
                                       *domainAssignment)
```

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;

TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to `trdc_master_t` in processor header file.
- regNum – Which register to configure, processor master can have more than one register for the MDAC configuration.
- domainAssignment – Pointer to the assignment structure.

```
void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const
                                         trdc_non_processor_domain_assignment_t
                                         *domainAssignment)
```

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to `trdc_master_t` in processor header file.
- domainAssignment – Pointer to the assignment structure.

```
static inline uint64_t TRDC_GetActiveMasterPidMap(TRDC_Type *base)
```

Gets the bit map of the bus master(s) that is(are) sourcing a PID register.

This function sets the non-processor master domain assignment as valid.

Parameters

- base – TRDC peripheral base address.

Returns

the bit map of the master(s). Bit 1 sets indicates bus master 1.

```
void TRDC_SetPid(TRDC_Type *base, uint8_t master, const trdc_pid_config_t *pidConfig)
```

Sets the current Process identifier(PID) for processor core.

Each processor has a corresponding process identifier (PID) which can be used to group tasks into different domains. Secure privileged software saves and restores the PID as part of any context switch. This data structure defines an array of 32-bit values, one per MDA

module, that define the PID. Since this register resource is only applicable to processor cores, the data structure is typically sparsely populated. The HWCFG[2-3] registers provide a bitmap of the implemented PIDn registers. This data structure is indexed using the corresponding MDA instance number. Depending on the operating clock domain of each DAC instance, there may be optional information stored in the corresponding PIDm register to properly implement the LK2 = 2 functionality.

Parameters

- base – TRDC peripheral base address.
- master – Which processor master to configure, refer to `trdc_master_t` in processor header file.
- pidConfig – Pointer to the configuration structure.

void TRDC_GetDefaultIDAUConfig(*trdc_idau_config_t* *idauConfiguration)

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```
config->lockSecureVTOR    = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU     = false;
config->lockNonsecureMPU  = false;
config->lockSAU           = false;
```

Parameters

- domainAssignment – Pointer to the configuration structure.

void TRDC_SetIDAU(TRDC_Type *base, const *trdc_idau_config_t* *idauConfiguration)

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
idauConfiguration.lockNonsecureMPU = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the configuration structure.

static inline void TRDC_EnableFlashLogicalWindow(TRDC_Type *base, bool enable)

Enables/disables the FLW(flash logical window) function.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

static inline void TRDC_LockFlashLogicalWindow(TRDC_Type *base)

Locks FLW registers. Once locked the registers can not be updated until next reset.

Parameters

- base – TRDC peripheral base address.

```
static inline uint32_t TRDC_GetFlashLogicalWindowPbase(TRDC_Type *base)
```

Gets the FLW physical base address.

Parameters

- base – TRDC peripheral base address.

Returns

Physical address of the FLW function.

```
static inline void TRDC_GetSetFlashLogicalWindowSize(TRDC_Type *base, uint16_t size)
```

Sets the FLW size.

Parameters

- base – TRDC peripheral base address.
- size – Size of the FLW in unit of 32k bytes.

```
void TRDC_GetDefaultFlashLogicalWindowConfig(trdc_flw_config_t *flwConfiguration)
```

Gets the default FLW(Flsh Logical Window) configuration.

```
config->blockCount  = false;
config->arrayBaseAddr = false;
config->lock        = false;
config->enable       = false;
```

Parameters

- flwConfiguration – Pointer to the configuration structure.

```
void TRDC_SetFlashLogicalWindow(TRDC_Type *base, const trdc_flw_config_t
                                *flwConfiguration)
```

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xFFFFFFFF;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- flwConfiguration – Pointer to the configuration structure.

```
status_t TRDC_GetAndClearFirstDomainError(TRDC_Type *base, trdc_domain_error_t *error)
```

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.

Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the `kStatus_NoData`.


```
status_t TRDC_GetAndClearFirstSpecificDomainError(TRDC_Type *base, trdc_domain_error_t
                                                    *error, uint8_t domainId)
```

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.
- domainId – The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the `kStatus_NoData`.

```
static inline void TRDC_SetMrcGlobalValid(TRDC_Type *base)
```

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

```
static inline uint8_t TRDC_GetMrcRegionNumber(TRDC_Type *base, uint8_t mrcIdx)
```

Gets the TRDC MRC(Memory Region Checkers) region number valid.

Parameters

- base – TRDC peripheral base address.

Returns

the region number of the given MRC instance

```
void TRDC_MrcSetMemoryAccessConfig(TRDC_Type *base, const
                                   trdc_memory_access_control_config_t *config, uint8_t
                                   mrcIdx, uint8_t regIdx)
```

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mrcIdx – MRC index.
- regIdx – Register number.

```
void TRDC_MrcEnableDomainNseUpdate(TRDC_Type *base, uint8_t mrcIdx, uint16_t  
                                   domianMask, bool enable)
```

Enables the update of the selected domians.

After the domians' update are enabled, their regions' NSE bits can be set or clear.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domianMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

```
void TRDC_MrcRegionNseSet(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

```
void TRDC_MrcRegionNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
```

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to clear.

```
void TRDC_MrcDomainNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)
```

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domians whose NSE bits to clear.

```
void TRDC_MrcSetRegionDescriptorConfig(TRDC_Type *base, const  
                                       trdc_mrc_region_descriptor_config_t *config)
```

Sets the configuration for one of the region descriptor per domain per MRC instnce.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to region descriptor configuration structure.

static inline void TRDC_SetMbcGlobalValid(TRDC_Type *base)

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

void TRDC_GetMbcHardwareConfig(TRDC_Type *base, *trdc_slave_memory_hardware_config_t* *config, uint8_t mbcIdx, uint8_t slvIdx)

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

void TRDC_MbcSetNseUpdateConfig(TRDC_Type *base, const *trdc_mbc_nse_update_config_t* *config, uint8_t mbcIdx)

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be updateby NSE set/clear.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

void TRDC_MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

void TRDC_MbcWordNseClear(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

void TRDC_MbcNseClearAll(TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slave)

Clears all configuration words' NSE bits of the selected domain and memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

```
void TRDC__MbcSetMemoryAccessConfig(TRDC_Type *base, const
                                     trdc_memory_access_control_config_t *config, uint8_t
                                     mbcIdx, uint8_t rgdIdx)
```

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC__SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

```
void TRDC__MbcSetMemoryBlockConfig(TRDC_Type *base, const
                                    trdc_mbc_memory_block_config_t *config)
```

Sets the configuration for one of the memory block per domain per MBC instance.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

```
enum _trdc_did_sel
```

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

Values:

enumerator kTRDC_DidMda

Use MDAn[2:0] as DID.

enumerator kTRDC_DidInput

Use the input DID (DID_in) as DID.

enumerator kTRDC_DidMdaAndInput

Use MDAn[2] concatenated with DID_in[1:0] as DID.

enumerator kTRDC_DidReserved

Reserved.

enum _trdc_secure_attr

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

Values:

enumerator kTRDC_ForceSecure

Force the bus attribute for this master to secure.

enumerator kTRDC_ForceNonSecure

Force the bus attribute for this master to non-secure.

enumerator kTRDC_MasterSecure

Use the bus master's secure/nonsecure attribute directly.

enumerator kTRDC_MasterSecure1

Use the bus master's secure/nonsecure attribute directly.

enum _trdc_pid_domain_hit_config

The configuration of domain hit evaluation of PID.

Values:

enumerator kTRDC_pidDomainHitNone0

No PID is included in the domain hit evaluation.

enumerator kTRDC_pidDomainHitNone1

No PID is included in the domain hit evaluation.

enumerator kTRDC_pidDomainHitInclusive

The PID is included in the domain hit evaluation when (PID & ~PIDM).

enumerator kTRDC_pidDomainHitExclusive

The PID is included in the domain hit evaluation when ~(PID & ~PIDM).

enum _trdc_privilege_attr

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

Values:

enumerator kTRDC_ForceUser

Force the bus attribute for this master to user.

enumerator kTRDC_ForcePrivilege

Force the bus attribute for this master to privileged.

enumerator kTRDC_MasterPrivilege

Use the bus master's attribute directly.

enumerator kTRDC_MasterPrivilege1

Use the bus master's attribute directly.

enum _trdc_pid_lock

PID lock configuration.

Values:

enumerator kTRDC_PidUnlocked0

The PID value can be updated by any secure privileged write.

enumerator kTRDC_PidUnlocked1

The PID value can be updated by any secure privileged write.

enumerator kTRDC_PidUnlocked2

The PID value can be updated by any secure privileged write from the bus master that first configured this register.

enumerator kTRDC_PidLocked

The PID value is locked until next reset.

enum _trdc_controller

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

Values:

enumerator kTRDC_MemBlockController0

Memory block checker 0.

enumerator kTRDC_MemBlockController1

Memory block checker 1.

enumerator kTRDC_MemBlockController2

Memory block checker 2.

enumerator kTRDC_MemBlockController3

Memory block checker 3.

enumerator kTRDC_MemRegionChecker0

Memory region checker 0.

enumerator kTRDC_MemRegionChecker1

Memory region checker 1.

enumerator kTRDC_MemRegionChecker2

Memory region checker 2.

enumerator kTRDC_MemRegionChecker3

Memory region checker 3.

enumerator kTRDC_MemRegionChecker4

Memory region checker 4.

enumerator kTRDC_MemRegionChecker5

Memory region checker 5.

enumerator kTRDC_MemRegionChecker6

Memory region checker 6.

enum _trdc_error_state

TRDC domain error state definition	TRDC_MBCn_DERR_W1[EST]	or
TRDC_MRCn_DERR_W1[EST].		

Values:

enumerator kTRDC_ErrorStateNone

No access violation detected.

enumerator kTRDC_ErrorStateNone1

No access violation detected.

enumerator kTRDC_ErrorStateSingle

Single access violation detected.

enumerator kTRDC_ErrorStateMulti

Multiple access violation detected.

enum _trdc_error_attr

TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

Values:

enumerator kTRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kTRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kTRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorSecurePrivilegeData
Secure privileged mode, data access.

enumerator kTRDC_ErrorNonSecureUserInst
NonSecure user mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecureUserData
NonSecure user mode, data access.

enumerator kTRDC_ErrorNonSecurePrivilegeInst
NonSecure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecurePrivilegeData
NonSecure privileged mode, data access.

enum _trdc_error_type

TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

Values:

enumerator kTRDC_ErrorTypeRead
Error occurs on read reference.

enumerator kTRDC_ErrorTypeWrite
Error occurs on write reference.

enum _trdc_region_descriptor

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

Values:

enumerator kTRDC_RegionDescriptor0
Region descriptor 0.

enumerator kTRDC_RegionDescriptor1
Region descriptor 1.

enumerator kTRDC_RegionDescriptor2
Region descriptor 2.

enumerator kTRDC_RegionDescriptor3
Region descriptor 3.

enumerator kTRDC_RegionDescriptor4
Region descriptor 4.

enumerator kTRDC_RegionDescriptor5
Region descriptor 5.

enumerator kTRDC_RegionDescriptor6

Region descriptor 6.

enumerator kTRDC_RegionDescriptor7

Region descriptor 7.

enumerator kTRDC_RegionDescriptor8

Region descriptor 8.

enumerator kTRDC_RegionDescriptor9

Region descriptor 9.

enumerator kTRDC_RegionDescriptor10

Region descriptor 10.

enumerator kTRDC_RegionDescriptor11

Region descriptor 11.

enumerator kTRDC_RegionDescriptor12

Region descriptor 12.

enumerator kTRDC_RegionDescriptor13

Region descriptor 13.

enumerator kTRDC_RegionDescriptor14

Region descriptor 14.

enumerator kTRDC_RegionDescriptor15

Region descriptor 15.

enum _trdc_MRC_domain

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

Values:

enumerator kTRDC_MrcDomain0

Domain 0.

enumerator kTRDC_MrcDomain1

Domain 1.

enumerator kTRDC_MrcDomain2

Domain 2.

enumerator kTRDC_MrcDomain3

Domain 3.

enumerator kTRDC_MrcDomain4

Domain 4.

enumerator kTRDC_MrcDomain5

Domain 5.

enumerator kTRDC_MrcDomain6

Domain 6.

enumerator kTRDC_MrcDomain7

Domain 7.

enumerator kTRDC_MrcDomain8

Domain 8.

enumerator kTRDC_MrcDomain9

Domain 9.

enumerator kTRDC_MrcDomain10

Domain 10.

enumerator kTRDC_MrcDomain11

Domain 11.

enumerator kTRDC_MrcDomain12

Domain 12.

enumerator kTRDC_MrcDomain13

Domain 13.

enumerator kTRDC_MrcDomain14

Domain 14.

enumerator kTRDC_MrcDomain15

Domain 15.

enum _trdc_MBC_domain

The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

Values:

enumerator kTRDC_MbcDomain0

Domain 0.

enumerator kTRDC_MbcDomain1

Domain 1.

enumerator kTRDC_MbcDomain2

Domain 2.

enumerator kTRDC_MbcDomain3

Domain 3.

enumerator kTRDC_MbcDomain4

Domain 4.

enumerator kTRDC_MbcDomain5

Domain 5.

enumerator kTRDC_MbcDomain6

Domain 6.

enumerator kTRDC_MbcDomain7

Domain 7.

enum _trdc_MBC_memory

The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

Values:

enumerator kTRDC_MbcSlaveMemory0

Memory 0.

enumerator kTRDC_MbcSlaveMemory1

Memory 1.

enumerator kTRDC_MbcSlaveMemory2

Memory 2.

enumerator kTRDC_MbcSlaveMemory3

Memory 3.

enum __trdc_MBC_bit

The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

Values:

enumerator kTRDC_MbcBit0

Bit 0.

enumerator kTRDC_MbcBit1

Bit 1.

enumerator kTRDC_MbcBit2

Bit 2.

enumerator kTRDC_MbcBit3

Bit 3.

enumerator kTRDC_MbcBit4

Bit 4.

enumerator kTRDC_MbcBit5

Bit 5.

enumerator kTRDC_MbcBit6

Bit 6.

enumerator kTRDC_MbcBit7

Bit 7.

enumerator kTRDC_MbcBit8

Bit 8.

enumerator kTRDC_MbcBit9

Bit 9.

enumerator kTRDC_MbcBit10

Bit 10.

enumerator kTRDC_MbcBit11

Bit 11.

enumerator kTRDC_MbcBit12

Bit 12.

enumerator kTRDC_MbcBit13

Bit 13.

enumerator kTRDC_MbcBit14

Bit 14.

enumerator kTRDC_MbcBit15

Bit 15.

enumerator kTRDC_MbcBit16

Bit 16.

enumerator `kTRDC_MbcBit17`

Bit 17.

enumerator `kTRDC_MbcBit18`

Bit 18.

enumerator `kTRDC_MbcBit19`

Bit 19.

enumerator `kTRDC_MbcBit20`

Bit 20.

enumerator `kTRDC_MbcBit21`

Bit 21.

enumerator `kTRDC_MbcBit22`

Bit 22.

enumerator `kTRDC_MbcBit23`

Bit 23.

enumerator `kTRDC_MbcBit24`

Bit 24.

enumerator `kTRDC_MbcBit25`

Bit 25.

enumerator `kTRDC_MbcBit26`

Bit 26.

enumerator `kTRDC_MbcBit27`

Bit 27.

enumerator `kTRDC_MbcBit28`

Bit 28.

enumerator `kTRDC_MbcBit29`

Bit 29.

enumerator `kTRDC_MbcBit30`

Bit 30.

enumerator `kTRDC_MbcBit31`

Bit 31.

typedef struct *_trdc_hardware_config* trdc_hardware_config_t

TRDC hardware configuration.

typedef struct *_trdc_slave_memory_hardware_config* trdc_slave_memory_hardware_config_t

Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *_trdc_did_sel* trdc_did_sel_t

TRDC domain ID select method, the register bit `TRDC_MDA_W0_0_DFMT0[DIDS]`, used for domain hit evaluation.

typedef enum *_trdc_secure_attr* trdc_secure_attr_t

TRDC secure attribute, the register bit `TRDC_MDA_W0_0_DFMT0[SA]`, used for bus master domain assignment.

typedef enum *_trdc_pid_domain_hit_config* trdc_pid_domain_hit_config_t

The configuration of domain hit evaluation of PID.

```
typedef struct _trdc_processor_domain_assignment trdc_processor_domain_assignment_t
    Domain assignment for the processor bus master.

typedef enum _trdc_privilege_attr trdc_privilege_attr_t
    TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-
    processor bus master domain assignment.

typedef struct _trdc_non_processor_domain_assignment
trdc_non_processor_domain_assignment_t
    Domain assignment for the non-processor bus master.

typedef enum _trdc_pid_lock trdc_pid_lock_t
    PID lock configuration.

typedef struct _trdc_pid_config trdc_pid_config_t
    Process identifier(PID) configuration for processor cores.

typedef struct _trdc_idau_config trdc_idau_config_t
    IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct _trdc_flw_config trdc_flw_config_t
    FLW(Flash Logical Window) configuration.

typedef enum _trdc_controller trdc_controller_t
    TRDC controller definition for domain error check. Each TRDC instance may have different
    MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

typedef enum _trdc_error_state trdc_error_state_t
    TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or
    TRDC_MRCn_DERR_W1[EST].

typedef enum _trdc_error_attr trdc_error_attr_t
    TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or
    TRDC_MRCn_DERR_W1[EATR].

typedef enum _trdc_error_type trdc_error_type_t
    TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

typedef struct _trdc_domain_error trdc_domain_error_t
    TRDC domain error definition.

typedef struct _trdc_memory_access_control_config trdc_memory_access_control_config_t
    Memory access control configuration for MBC/MRC.

typedef struct _trdc_mrc_region_descriptor_config trdc_mrc_region_descriptor_config_t
    The configuration of each region descriptor per domain per MRC instance.

typedef struct _trdc_mbc_nse_update_config trdc_mbc_nse_update_config_t
    The configuration of MBC NSE update.

typedef struct _trdc_mbc_memory_block_config trdc_mbc_memory_block_config_t
    The configuration of each memory block per domain per MBC instance.

FSL_TRDC_DRIVER_VERSION

struct _trdc_hardware_config
    #include <fsl_trdc.h> TRDC hardware configuration.
```

Public Members

uint8_t masterNumber
Number of bus masters.

uint8_t domainNumber
Number of domains.

uint8_t mbcNumber
Number of MBCs.

uint8_t mrcNumber
Number of MRCs.

struct _trdc_slave_memory_hardware_config
#include <fsl_trdc.h> Hardware configuration of the two slave memories within each MBC(memory block checker).

Public Members

uint32_t blockNum
Number of blocks.

uint32_t blockSize
Block size.

struct _trdc_processor_domain_assignment
#include <fsl_trdc.h> Domain assignment for the processor bus master.

Public Members

uint32_t domainId
Domain ID.

uint32_t domainIdSelect
Domain ID select method, see trdc_did_sel_t.

uint32_t pidDomainHitConfig
The configuration of the domain hit evaluation for PID, see trdc_pid_domain_hit_config_t.

uint32_t pidMask
The mask combined with PID, so multiple PID can be included as part of the domain hit determination. Set to 0 to disable.

uint32_t secureAttr
Secure attribute, see trdc_secure_attr_t.

uint32_t pid
The process identifier, combined with pidMask to form the domain hit determination.

uint32_t __pad0__
Reserved.

uint32_t lock
Lock the register.

uint32_t __pad1__
Reserved.

struct _trdc_non_processor_domain_assignment
#include <fsl_trdc.h> Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId

Domain ID.

uint32_t privilegeAttr

Privileged attribute, see trdc_privilege_attr_t.

uint32_t secureAttr

Secure attribute, see trdc_secure_attr_t.

uint32_t bypassDomainId

Bypass domain ID.

uint32_t __pad0__

Reserved.

uint32_t lock

Lock the register.

uint32_t __pad1__

Reserved.

struct _trdc_pid_config

#include <fsl_trdc.h> Process identifier(PID) configuration for processor cores.

Public Members

uint32_t pid

The process identifier of the executing task. The highest bit can be used to define secure/nonsecure attribute of the task.

uint32_t __pad0__

Reserved.

uint32_t lock

How to lock the register, see trdc_pid_lock_t.

uint32_t __pad1__

Reserved.

struct _trdc_idau_config

#include <fsl_trdc.h> IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

Public Members

uint32_t __pad0__

Reserved.

uint32_t lockSecureVTOR

Disable writes to secure VTOR(Vector Table Offset Register).

uint32_t lockNonsecureVTOR

Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32_t lockSecureMPU

Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32_t lockNonsecureMPU

Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32_t lockSAU

Disable writes to SAU(Security Attribution Unit) registers.

uint32_t __pad1__

Reserved.

struct __trdc_flw_config

#include <fsl_trdc.h> FLW(Flash Logical Window) configuration.

Public Members

uint16_t blockCount

Block count of the Flash Logic Window in 32KByte blocks.

uint32_t arrayBaseAddr

Flash array base address of the Flash Logical Window.

bool lock

Disable writes to FLW registers.

bool enable

Enable FLW function.

struct __trdc_domain_error

#include <fsl_trdc.h> TRDC domain error definition.

Public Members

trdc_controller_t controller

Which controller captured access violation.

uint32_t address

Access address that generated access violation.

trdc_error_state_t errorState

Error state.

trdc_error_attr_t errorAttr

Error attribute.

trdc_error_type_t errorType

Error type.

uint8_t errorPort

Error port.

uint8_t domainId

Domain ID.

uint8_t slaveMemoryIdx

The slave memory index. Only apply when violation in MBC.

struct __trdc_memory_access_control_config

#include <fsl_trdc.h> Memory access control configuration for MBC/MRC.

Public Members

uint32_t nonsecureUsrX

Allow nonsecure user execute access.

uint32_t nonsecureUsrW

Allow nonsecure user write access.

uint32_t nonsecureUsrR

Allow nonsecure user read access.

uint32_t __pad0__

Reserved.

uint32_t nonsecurePrivX

Allow nonsecure privilege execute access.

uint32_t nonsecurePrivW

Allow nonsecure privilege write access.

uint32_t nonsecurePrivR

Allow nonsecure privilege read access.

uint32_t __pad1__

Reserved.

uint32_t secureUsrX

Allow secure user execute access.

uint32_t secureUsrW

Allow secure user write access.

uint32_t secureUsrR

Allow secure user read access.

uint32_t __pad2__

Reserved.

uint32_t securePrivX

Allownonsecure privilege execute access.

uint32_t securePrivW

Allownonsecure privilege write access.

uint32_t securePrivR

Allownonsecure privilege read access.

uint32_t __pad3__

Reserved.

uint32_t lock

Lock the configuration until next reset, only apply to access control register 0.

struct _trdc_mrc_region_descriptor_config

#include <fsl_trdc.h> The configuration of each region descriptor per domain per MRC instance.

Public Members

uint8_t memoryAccessControlSelect

Select one of the 8 access control policies for this region, for access cotrol policies see trdc_memory_access_control_config_t.

`uint32_t startAddr`
Physical start address.

`bool valid`
Lock the register.

`bool nseEnable`
Enable non-secure accesses and disable secure accesses.

`uint32_t endAddr`
Physical start address.

`uint8_t mrcIdx`
The index of the MRC for this configuration to take effect.

`uint8_t domainIdx`
The index of the domain for this configuration to take effect.

`uint8_t regionIdx`
The index of the region for this configuration to take effect.

`struct _trdc_mbc_nse_update_config`
#include <fsl_trdc.h> The configuration of MBC NSE update.

Public Members

`uint32_t __pad0__`
Reserved.

`uint32_t wordIdx`
MBC configuration word index to be updated.

`uint32_t __pad1__`
Reserved.

`uint32_t memorySelect`
Bit mask of the selected memory to be updated. `_trdc_MBC_memory`.

`uint32_t __pad2__`
Reserved.

`uint32_t domainSelect`
Bit mask of the selected domain to be updated. `_trdc_MBC_domain`.

`uint32_t __pad3__`
Reserved.

`uint32_t autoIncrement`
Whether to increment the word index after current word is updated using this configuration.

`struct _trdc_mbc_memory_block_config`
#include <fsl_trdc.h> The configuration of each memory block per domain per MBC instance.

Public Members

`uint32_t memoryAccessControlSelect`
Select one of the 8 access control policies for this memory block, for access control policies see `trdc_memory_access_control_config_t`.

`uint32_t nseEnable`

Enable non-secure accesses and disable secure accesses.

`uint32_t mbcIdx`

The index of the MBC for this configuration to take effect.

`uint32_t domainIdx`

The index of the domain for this configuration to take effect.

`uint32_t slaveMemoryIdx`

The index of the slave memory for this configuration to take effect.

`uint32_t memoryBlockIdx`

The index of the memory block for this configuration to take effect.

2.85 Trdc_core

`typedef struct _TRDC_General_Type TRDC_General_Type`

TRDC general configuration register definition.

`typedef struct _TRDC_FLW_Type TRDC_FLW_Type`

TRDC flash logical control register definition.

`typedef struct _TRDC_DomainError_Type TRDC_DomainError_Type`

TRDC domain error register definition.

`typedef struct _TRDC_DomainAssignment_Type TRDC_DomainAssignment_Type`

TRDC master domain assignment register definition.

`typedef struct _TRDC_MBC_Type TRDC_MBC_Type`

TRDC MBC control register definition.

`typedef struct _TRDC_MRC_Type TRDC_MRC_Type`

TRDC MRC control register definition. `MRC_DOM0_RGD_W[region][word]`.

`TRDC_GENERAL_BASE(base)`

TRDC base address convert macro.

`TRDC_FLW_BASE(base)`

`TRDC_DOMAIN_ERROR_BASE(base)`

`TRDC_DOMAIN_ASSIGNMENT_BASE(base)`

`TRDC_MBC_BASE(base, instance)`

`TRDC_MRC_BASE(base, instance)`

`struct _TRDC_General_Type`

`#include <fsl_trdc_core.h>` TRDC general configuration register definition.

Public Members

`__IO uint32_t TRDC_CR`

TRDC Register, offset: 0x0

`__I uint32_t TRDC_HWCFG0`

TRDC Hardware Configuration Register 0, offset: 0xF0

```

__IO uint32_t TRDC_HWCFG1
    TRDC Hardware Configuration Register 1, offset: 0xF4
__IO uint32_t TRDC_HWCFG2
    TRDC Hardware Configuration Register 2, offset: 0xF8
__IO uint32_t TRDC_HWCFG3
    TRDC Hardware Configuration Register 3, offset: 0xFC
__IO uint8_t DACFG [8]
    Domain Assignment Configuration Register, array offset: 0x100, array step: 0x1
__IO uint32_t TRDC_IDAU_CR
    TRDC IDAU Control Register, offset: 0x1C0

```

```
struct _TRDC_FLW_Type
```

```
#include <fsl_trdc_core.h> TRDC flash logical control register definition.
```

Public Members

```

__IO uint32_t TRDC_FLW_CTL
    TRDC FLW Control, offset: 0x1E0
__IO uint32_t TRDC_FLW_PBASE
    TRDC FLW Physical Base, offset: 0x1E4
__IO uint32_t TRDC_FLW_ABASE
    TRDC FLW Array Base, offset: 0x1E8
__IO uint32_t TRDC_FLW_BCNT
    TRDC FLW Block Count, offset: 0x1EC

```

```
struct _TRDC_DomainError_Type
```

```
#include <fsl_trdc_core.h> TRDC domain error register definition.
```

Public Members

```

__IO uint32_t TRDC_FDID
    TRDC Fault Domain ID, offset: 0x1FC
__IO uint32_t TRDC_DERRLOC [16]
    TRDC Domain Error Location Register, array offset: 0x200, array step: 0x4

```

```
struct _TRDC_DomainAssignment_Type
```

```
#include <fsl_trdc_core.h> TRDC master domain assignment register definition.
```

Public Members

```

__IO uint32_t PID [8]
    Process Identifier, array offset: 0x700, array step: 0x4

```

```
struct _TRDC_MBC_Type
```

```
#include <fsl_trdc_core.h> TRDC MBC control register definition.
```

Public Members

- ___I uint32_t MBC_MEM_GLBCFG [4]
MBC Global Configuration Register, array offset: 0x10000, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_NSE_BLK_INDEX
MBC NonSecure Enable Block Index, array offset: 0x10010, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_SET
MBC NonSecure Enable Block Set, array offset: 0x10014, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_CLR
MBC NonSecure Enable Block Clear, array offset: 0x10018, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_CLR_ALL
MBC NonSecure Enable Block Clear All, array offset: 0x1001C, array step: 0x2000
- ___IO uint32_t MBC_MEMN_GLBAC [8]
MBC Global Access Control, array offset: 0x10020, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10340, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM1_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10380, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10640, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM3_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10740, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10780, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10840, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10940, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10980, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109F0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM5_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10A40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10D80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DD0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM6_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10E40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10F40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10F80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111C8, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM8_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11340, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11380, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM10_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11640, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11740, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11780, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117A0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117A8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117C8, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117D0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM11_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117F0, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11840, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11940, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11980, array step: index*0x2000, index2*0x4

___IO uint32_t MBC_DOM12_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119A0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM12_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11A40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11D80, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11DA0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11DA8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11DC8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11DD0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11DF0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11E40, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11F40, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11F80, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11FA0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11FA8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11FC8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11FD0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11FF0, array step: index*0x2000, index2*0x4

struct _TRDC_MRC_Type

#include <fsl_trdc_core.h> TRDC MRC control register definition.
MRC_DOM0_RGD_W[region][word].

Public Members

__IO uint32_t MRC_GLBCFG
MRC Global Configuration Register, array offset: 0x14000, array step: 0x1000

__IO uint32_t MRC_NSE_RGN_INDIRECT
MRC NonSecure Enable Region Indirect, array offset: 0x14010, array step: 0x1000

__IO uint32_t MRC_NSE_RGN_SET
MRC NonSecure Enable Region Set, array offset: 0x14014, array step: 0x1000

__IO uint32_t MRC_NSE_RGN_CLR
MRC NonSecure Enable Region Clear, array offset: 0x14018, array step: 0x1000

__IO uint32_t MRC_NSE_RGN_CLR_ALL
MRC NonSecure Enable Region Clear All, array offset: 0x1401C, array step: 0x1000

__IO uint32_t MRC_GLBAC [8]
MRC Global Access Control, array offset: 0x14020, array step: index*0x1000, index2*0x4

__IO uint32_t MRC_DOM0_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14040, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM0_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x140C0, array step: 0x1000

__IO uint32_t MRC_DOM1_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14140, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM1_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x141C0, array step: 0x1000

__IO uint32_t MRC_DOM2_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14240, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM2_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x142C0, array step: 0x1000

__IO uint32_t MRC_DOM3_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14340, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM3_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x143C0, array step: 0x1000

__IO uint32_t MRC_DOM4_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14440, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM4_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x144C0, array step: 0x1000

__IO uint32_t MRC_DOM5_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14540, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM5_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x145C0, array step: 0x1000

__IO uint32_t MRC_DOM6_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14640, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM6_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x146C0, array step: 0x1000

__IO uint32_t MRC_DOM7_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14740, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM7_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x147C0, array step: 0x1000

__IO uint32_t MRC_DOM8_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14840, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM8_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x148C0, array step: 0x1000

__IO uint32_t MRC_DOM9_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14940, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM9_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x149C0, array step: 0x1000

__IO uint32_t MRC_DOM10_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14A40, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM10_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14AC0, array step: 0x1000

__IO uint32_t MRC_DOM11_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14B40, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM11_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14BC0, array step: 0x1000

__IO uint32_t MRC_DOM12_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14C40, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM12_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14CC0, array step: 0x1000

__IO uint32_t MRC_DOM13_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14D40, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM13_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14DC0, array step: 0x1000

__IO uint32_t MRC_DOM14_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14E40, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM14_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14EC0, array step: 0x1000

__IO uint32_t MRC_DOM15_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14F40, array step: index*0x1000, index2*0x8, index3*0x4

__IO uint32_t MRC_DOM15_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14FC0, array step: 0x1000
struct MBC_DERR

Public Members

__I uint32_t W0
MBC Domain Error Word0 Register, array offset: 0x400, array step: 0x10
__I uint32_t W1
MBC Domain Error Word1 Register, array offset: 0x404, array step: 0x10
__O uint32_t W3
MBC Domain Error Word3 Register, array offset: 0x40C, array step: 0x10
struct MRC_DERR

Public Members

__I uint32_t W0
MRC Domain Error Word0 Register, array offset: 0x480, array step: 0x10
__I uint32_t W1
MRC Domain Error Word1 Register, array offset: 0x484, array step: 0x10
__O uint32_t W3
MRC Domain Error Word3 Register, array offset: 0x48C, array step: 0x10
union __unnamed69__

Public Members

struct *_TRDC_DomainAssignment_Type* MDA_DFMT0[8]
struct *_TRDC_DomainAssignment_Type* MDA_DFMT1[8]
struct MDA_DFMT0

Public Members

__IO uint32_t MDA_W_DFMT0 [8]
DAC Master Domain Assignment Register, array offset: 0x800, array step: index*0x20,
index2*0x4
struct MDA_DFMT1

Public Members

__IO uint32_t MDA_W_DFMT1 [1]
DAC Master Domain Assignment Register, array offset: 0x800, array step: index*0x20,
index2*0x4

2.86 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.

2.87 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.

2.88 WUU: Wakeup Unit driver

```
void WUU_SetExternalWakeUpPinsConfig(WUU_Type *base, uint8_t pinIndex, const
                                     wuu_external_wakeup_pin_config_t *config)
```

Enables and Configs External WakeUp Pins.

This function enables/disables the external pin as wakeup input. What's more this function configs pins options, including edge detection wakeup event and operate mode.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin. See Reference Manual for the details.
- config – Pointer to `wuu_external_wakeup_pin_config_t` structure.

```
void WUU_ClearExternalWakeupPinsConfig(WUU_Type *base, uint8_t pinIndex)
```

Disable and clear external wakeup pin settings.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin.

```
static inline uint32_t WUU_GetExternalWakeUpPinsFlag(WUU_Type *base)
```

Gets External Wakeup pin flags.

This function return the external wakeup pin flags.

Parameters

- base – WUU peripheral base address.

Returns

Wakeup flags for all external wakeup pins.

```
static inline void WUU_ClearExternalWakeUpPinsFlag(WUU_Type *base, uint32_t mask)
```

Clears External WakeUp Pin flags.

This function clears external wakeup pins flags based on the mask.

Parameters

- base – WUU peripheral base address.
- mask – The mask of Wakeup pin index to be cleared.

```
void WUU_SetInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                         wuu_internal_wakeup_module_event_t event)
```

Config Internal modules' event as the wake up sources.

This function configs the internal modules event as the wake up sources.

Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – Select interrupt or DMA/Trigger of the internal module as the wake up source.

```
void WUU_ClearInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                           wuu_internal_wakeup_module_event_t event)
```

Disable an on-chip internal modules' event as the wakeup sources.

Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – The event(interrupt or DMA/trigger) of the internal module to disable.

```
void WUU_SetPinFilterConfig(WUU_Type *base, uint8_t filterIndex, const
                           wuu_pin_filter_config_t *config)
```

Configs and Enables Pin filters.

This function configs Pin filter, including pin select, filter operate mode filter wakeup event and filter edge detection.

Parameters

- base – WUU peripheral base address.
- filterIndex – The index of the pin filter.
- config – Pointer to wuu_pin_filter_config_t structure.

```
bool WUU_GetPinFilterFlag(WUU_Type *base, uint8_t filterIndex)
```

Gets the pin filter configuration.

This function gets the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

```
void WUU_ClearPinFilterFlag(WUU_Type *base, uint8_t filterIndex)
```

Clears the pin filter configuration.

This function clears the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

```
bool WUU_GetExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)
```

brief Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0. return True if the specific pin is a wakeup source.

```
void WUU_ClearExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)
```

brief Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0.

```
FSL_WUU_DRIVER_VERSION
```

Defines WUU driver version 2.4.1.

```
enum _wuu_external_pin_edge_detection
```

External WakeUp pin edge detection enumeration.

Values:

enumerator kWUU_ExternalPinDisable

External input Pin disabled as wake up input.

enumerator kWUU_ExternalPinRisingEdge

External input Pin enabled with the rising edge detection.

enumerator kWUU_ExternalPinFallingEdge

External input Pin enabled with the falling edge detection.

enumerator kWUU_ExternalPinAnyEdge

External input Pin enabled with any change detection.

enum __wu__external_wakeup_pin_event

External input wake up pin event enumeration.

Values:

enumerator kWUU_ExternalPinInterrupt

External input Pin configured as interrupt.

enumerator kWUU_ExternalPinDMARequest

External input Pin configured as DMA request.

enumerator kWUU_ExternalPinTriggerEvent

External input Pin configured as Trigger event.

enum __wu__external_wakeup_pin_mode

External input wake up pin mode enumeration.

Values:

enumerator kWUU_ExternalPinActiveDSPD

External input Pin is active only during Deep Sleep/Power Down Mode. NOTE: This enumerations has been deprecated, please switch to kWUU_ExternalPinActiveLowLeakage.

enumerator kWUU_ExternalPinActiveLowLeakageMode

External input Pin is active only during low-leakage power modes.

enumerator kWUU_ExternalPinActiveAlways

External input Pin is active during all power modes.

enum __wu__internal_wakeup_module_event

Internal module wake up event enumeration.

Values:

enumerator kWUU_InternalModuleInterrupt

Internal modules' interrupt as a wakeup source.

enumerator kWUU_InternalModuleDMATrigger

Internal modules' DMA/Trigger as a wakeup source.

enum __wu__filter_edge

Pin filter edge enumeration.

Values:

enumerator kWUU_FilterDisabled

Filter disabled.

enumerator kWUU_FilterPosedgeEnable

Filter posedge detect enabled.

enumerator kWUU_FilterNegedgeEnable

Filter negedge detect enabled.

```

    enumerator kWUU_FilterAnyEdge
        Filter any edge detect enabled.

enum _wu_filter_event
    Pin Filter event enumeration.

    Values:

    enumerator kWUU_FilterInterrupt
        Filter output configured as interrupt.

    enumerator kWUU_FilterDMARequest
        Filter output configured as DMA request.

    enumerator kWUU_FilterTriggerEvent
        Filter output configured as Trigger event.

enum _wu_filter_mode
    Pin filter mode enumeration.

    Values:

    enumerator kWUU_FilterActiveDSPD
        External input pin filter is active only during Deep Sleep/Power Down
        Mode. NOTE: This enumerations has been deprecated, please switch to
        kWUU_FilterActiveLowLeakage.

    enumerator kWUU_FilterActiveLowLeakageMode
        External input pin filter is active only during low-leakage power modes.

    enumerator kWUU_FilterActiveAlways
        External input Pin filter is active during all power modes.

typedef enum _wu_external_pin_edge_detection wu_external_pin_edge_detection_t
    External WakeUp pin edge detection enumeration.

typedef enum _wu_external_wakeup_pin_event wu_external_wakeup_pin_event_t
    External input wake up pin event enumeration.

typedef enum _wu_external_wakeup_pin_mode wu_external_wakeup_pin_mode_t
    External input wake up pin mode enumeration.

typedef enum _wu_internal_wakeup_module_event wu_internal_wakeup_module_event_t
    Internal module wake up event enumeration.

typedef enum _wu_filter_edge wu_filter_edge_t
    Pin filter edge enumeration.

typedef enum _wu_filter_event wu_filter_event_t
    Pin Filter event enumeration.

typedef enum _wu_filter_mode wu_filter_mode_t
    Pin filter mode enumeration.

typedef struct _wu_external_wakeup_pin_config wu_external_wakeup_pin_config_t
    External WakeUp pin configuration.

typedef struct _wu_pin_filter_config wu_pin_filter_config_t
    Pin Filter configuration.

struct _wu_external_wakeup_pin_config
    #include <fsl_wuu.h> External WakeUp pin configuration.

```

Public Members

wuu_external_pin_edge_detection_t edge

External Input pin edge detection.

wuu_external_wakeup_pin_event_t event

External Input wakeup Pin event

wuu_external_wakeup_pin_mode_t mode

External Input wakeup Pin operate mode.

struct *_wuu_pin_filter_config*

#include <fsl_wuu.h> Pin Filter configuration.

Public Members

uint32_t pinIndex

The index of wakeup pin to be muxxed into filter.

wuu_filter_edge_t edge

The edge of the pin digital filter.

wuu_filter_event_t event

The event of the filter output.

wuu_filter_mode_t mode

The mode of the filter operate.

2.89 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 = 0xFFFFFFFU;
config->timeoutValue = 0xFFFFFFFU;
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 `0xFFFFFFFF` (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](#)

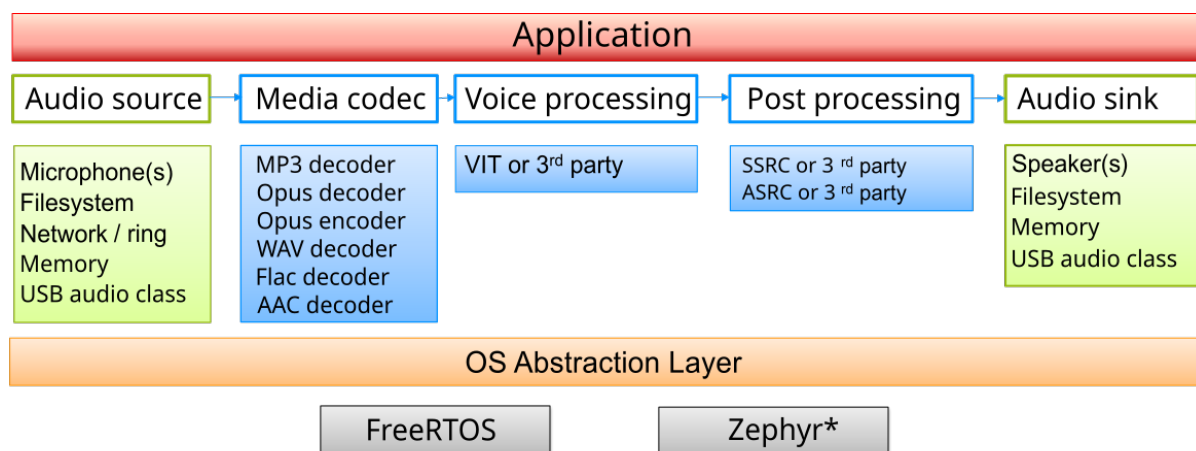
Visit [Maestro - Documentation](#) to review details on the contents in this sub-repo.

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

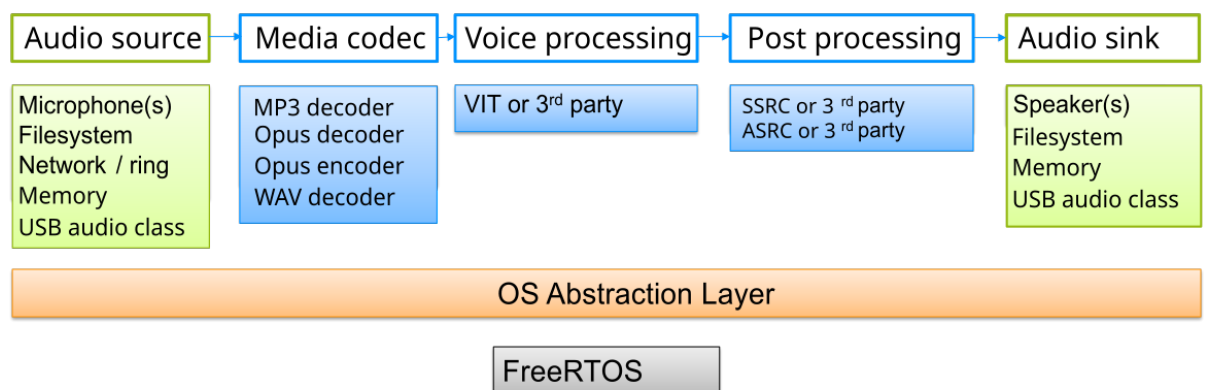
File structure

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/utlis	Helper utilities utilized by Maestro
docs	Generated documentation
doxygen	Documentation sources
components	Glue for audio libraries, so they can be used in elements
tests	Maestro tests
zephyr/	Zephyr related files
zephyr/samples/	Zephyr samples
zephyr/tests/	Zephyr tests
zephyr/audioTracks/	Audio tracks for testing
zephyr/wrappers/	Zephyr NXP SDK Wrappers
zephyr/doc/	Zephyr documentation configuration for Sphinx
zephyr/scripts/	Zephyr helper scripts, mostly for testing

Maestro Audio Framework Programmer's Guide

Introduction Maestro audio framework provides instruments for playback and capture of different audio streams. In order to do that the framework uses API for creating various audio and voice pipelines with the support of media and track information. This document describes the framework in its detail, and the usage of API for pipeline creation using different elements. The framework needs an operating system in order to create different tasks for audio processing and communication with the application.

Architecture overview A high-level block diagram of the streamer used in Maestro is shown below. An element is the most important class of objects in the streamer (see `streamer_element.c`). A chain of elements will be created and linked together when a *pipeline* is created. Data flows through this chain of elements in form of data buffers. An element has one specific function, which can be the reading of data from a file, decoding of this data, or outputting this data to a sink device. By chaining together several such elements, a pipeline is created that can do a specific task, for example, the playback.



Pipeline

The pipeline is created within the `streamer_create` API using the `streamer_create_pipeline` call. In the example applications provided in the MCUXpresso SDK the pipeline is created in the `app_streamer.c` file. In order to create a pipeline user needs to provide a `PipelineElements` structure consisting of array of element indexes `ElementIndex` and the number of elements in the pipeline. Then the pipeline is built automatically and user can specify the properties of the elements using the `streamer_set_property` API. All the element properties can be found in the `streamer_element_properties.h` file.

The streamer can handle up to two pipelines within a single task. The first pipeline with index 0 can be created using the `streamer_create` function as described above. Then the `streamer_create_pipeline` function should be used to create the second pipeline (pipeline with index 1). Both pipelines are processed sequentially, so after the first pipeline is processed, the second pipeline is processed.

After the pipeline is successfully created, all elements and entire pipeline are in `STATE_NULL` state. A user can start the streamer by setting the pipeline state to `STATE_PLAYING` using the `streamer_set_state` function. The pipeline can also be paused or stopped using the same function. Use the `STATE_PAUSED` to pause and use `STATE_NULL` to stop. The function changes the state of each element that is in the pipeline in turn, and after all the elements have obtained the desired state, the state of entire pipeline is changed.

Elements The current version of the Maestro framework supports several types of elements (`StreamElementType`). In each pipeline should be used one source element (elements with the `_SRC` suffix) and one sink element (elements with the `_SINK` suffix). A decoder, encoder or `audio_proc` element can be connected between these two elements. The `audio_proc` element can be used more than once within the same pipeline.

Each element type (`StreamElementType`) has several functions that are determined by a unique element index (`ElementIndex`). These indexes are used to create a pipeline, and each element index can only be used once in the same pipeline. The `type_lookup_table` shows which `StreamElementType` supports which `ElementIndex`.

Each element index (`ElementIndex`) has its own properties and a list of these properties can be found in the `streamer_element_properties.h` file. These properties are divided into groups and each group is identified by a property mask (e.g. for speaker it is `PROP_SPEAKER_MASK`). Then the `property_lookup_table` in the `streamer_msg.c` file determines which property group relates to which element index (`ElementIndex`). When an element is created and added to the pipeline, its properties are set to their default values. Default values can be seen in the initialization function of a particular element. The initialization functions are specified in the `element_list` array in the `streamer_element.c` file (e.g. for the `audio_proc` element it is the `audio_proc_init_element` function). The user can get the value of the property using the `streamer_get_property` function or change its value using the `streamer_set_property` function.

The source code of the elements can be found in the `middleware\audio_voice\maestro\src\elements\` folder.

Add a new element type The user can add a new element type (`StreamElementType`) to the Maestro audio framework. For this, the following steps need to be done.

- Add a new element type to the `StreamElementType` enum type in the `streamer_api.h`.
- Create a new `*.c` and `*.h` files for the new element type in the `middleware\audio_voice\maestro\src\elements\` folder. All necessary structures and functions (functions for src pads, sink pads and element itself) needs to be defined in these files. Inspiration can be found in other elements.
- Link the initialization function to the element type in the `element_list` array in the `streamer_element.c` file. To do this, a new definition that enables the element needs to be created (e.g. there is a `STREAMER_ENABLE_AUDIO_PROC` definition for the `audio_proc` element).

- Associate the newly created element type with an element index (ElementIndex) by adding a new pair to the `type_lookup_table` in the `streamer.c` file.
- If the user wants to use the newly created element in an application, the definition that enables the element must be defined at the project level.

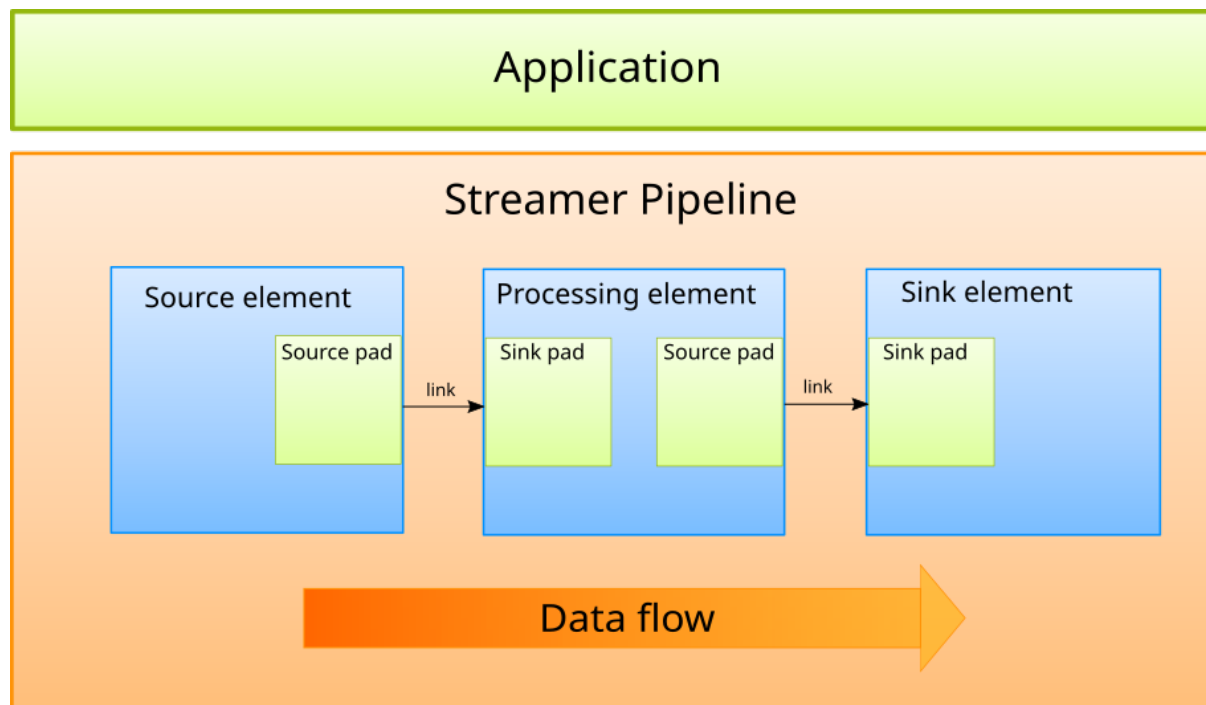
Mostly the user doesn't need to create a new element type, but just create an element index.

Add a new element index To create a new element index in the Maestro audio framework, follow these steps:

- Add a new element index to the `ElementIndex` enum type in the `streamer_api.h`.
- Create the required properties for the newly created element index in the `streamer_element_properties.h` file.
- Associate the newly created property group with newly created element index by adding a new pair to the `property_lookup_table` in the `streamer_msg.c` file.
- Associate the newly created element index with an element type (`StreamElementType`) by adding a new pair to the `type_lookup_table` in the `streamer.c` file.
- Add support for the created properties to functions of the associated element type. These functions are defined in files that correspond to a particular element type. The files are located in the `middleware\audio_voice\maestro\src\elements\` folder.

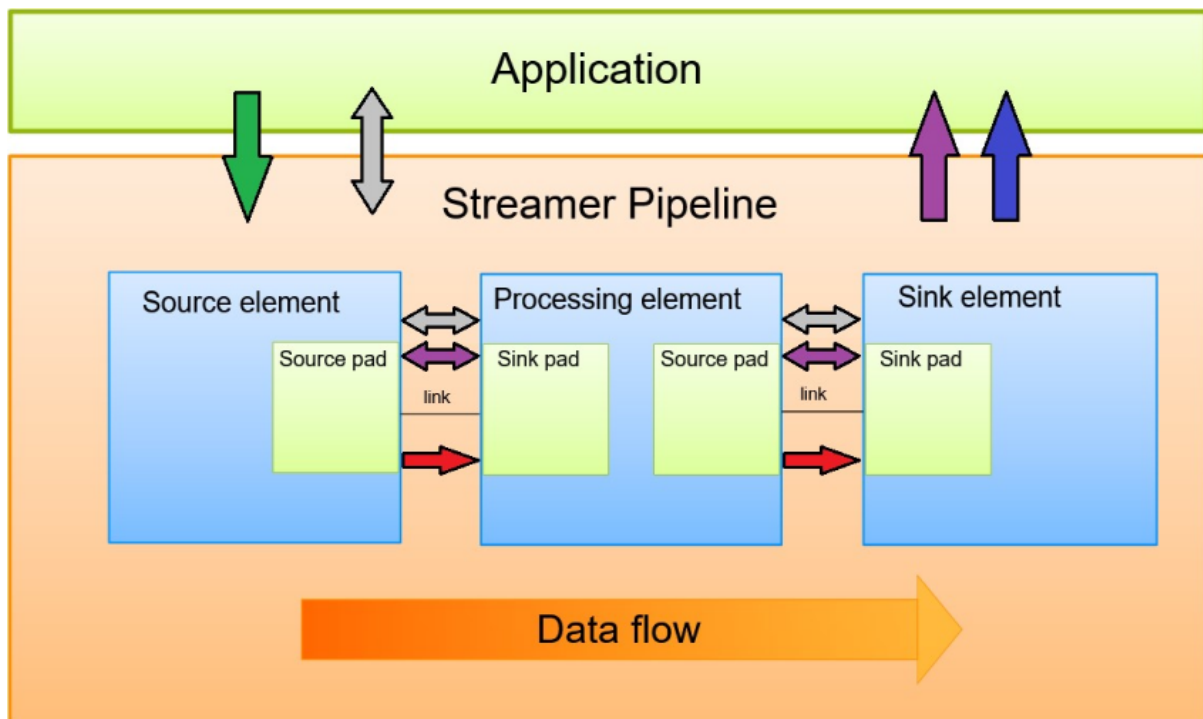
It is important to know that each element type (`StreamElementType`) can be associated with more than one element index (`ElementIndex`), but each element index (`ElementIndex`) can be associated with only one element type (`StreamElementType`).

Pads Pads are elements' inputs and outputs. A pad can be viewed as a “plug” or “port” on an element where links may be made with other elements, and through which data can flow to or from those elements. Data flows out of an element through a source pad, and elements accept incoming data through a sink pad. Source and sink elements have only source and sink pads, respectively. For detailed information about pads, please see the API reference from `pad.c`.



Internal communication The streamer (the core of the framework) provides several mechanisms for communication and data exchange between the application, a pipeline, and pipeline elements:

- Buffers are objects for passing streaming data between elements in the pipeline. Buffers always travel from sources to sinks (downstream).
- Messages are objects sent from the application to the streamer task to construct, configure, and control a streamer pipeline.
- Callbacks are used to transmit information such as errors, tags, state changes, etc. from the pipeline and elements to the application.
- Events are objects sent between elements. Events can travel upstream and downstream. Events may also be sent to the application.
- Queries allow applications to request information such as duration or current playback position from the pipeline. Elements can also use queries to request information from their peer elements (such as the file size or duration). They can be used both ways within a pipeline, but upstream queries are more common.



Decoders and encoders Maestro framework uses a common codec interface for decoding purposes and a common encoder interface for encoding. Those interfaces encapsulate the usage of specific codecs. Reference codecs are available in audio-voice-components repository which should be in `\middleware\audio_voice\components\` folder.

Common codec interface The Common Codec Interface is the intended interface for all used **decoders**. The framework will integrate a CCI decoder element into the streamer to interface with all decoders.

Using the CCI to interface with Metadata

- `cci_extract_meta_data` must be called before any other Codec Interface APIs. This API extracts the metadata information of the codec and fills this information in the

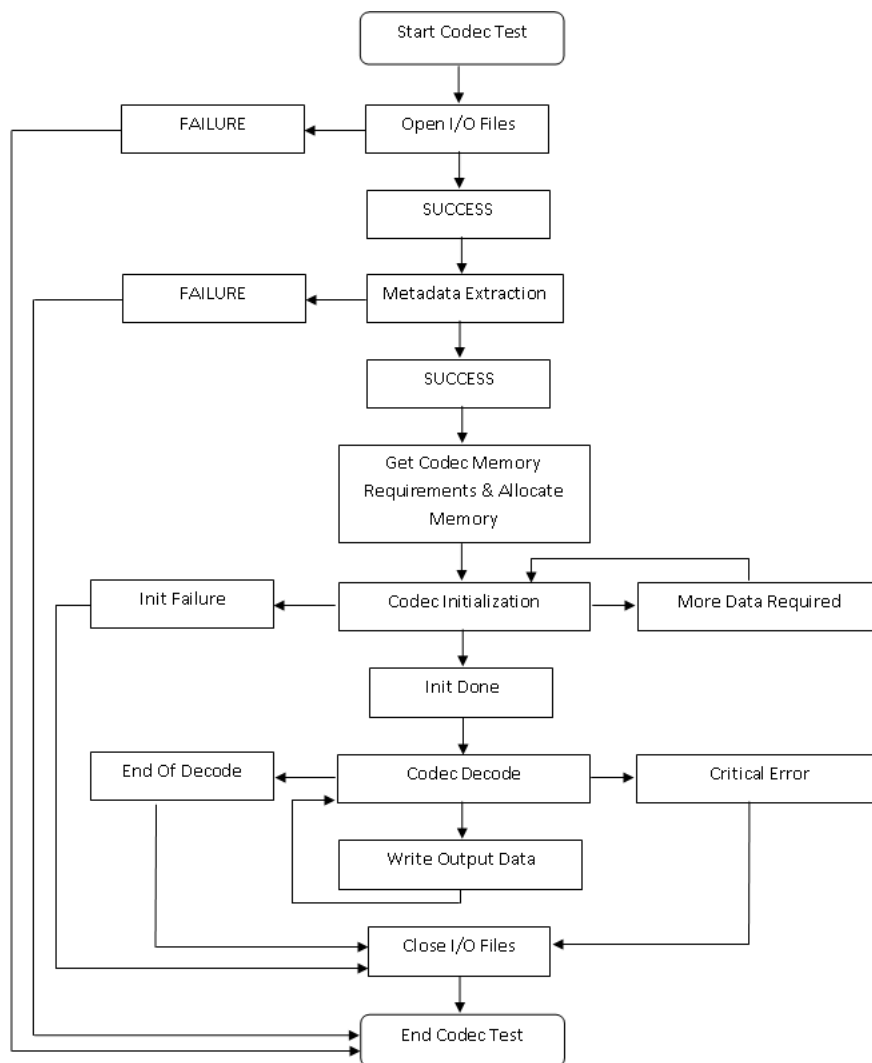
file_meta_data_t structure. The file_meta_data_t structure must be allocated by the application.

- This function first extracts the input file extension and based on that it calls the specific codec's metadata extraction function. If it finds an invalid extension or unsupported extension then it returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.
- If this API finds the valid metadata then it returns with META_DATA_FOUND code. If this API does not find any metadata information then it returns with META_DATA_NOT_FOUND code. It also returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.

Using the CCI to interface with Decoders

- `codec_get_mem_info` gets the memory requirement based on the specific decoder stream type. It returns the size in bytes of the specific codec. The user of the decoders must allocate memory of this size and this memory is used by the initialization API. The user or application must pass this allocated memory pointer to the init API.
- `codec_init` must be called before the codec's decode API. This API calls the codec-specific initialization function based on the codec stream type. This API allocates the memory to the codec main structure and also initializes the codec main structure parameters. It also registers the call back functions to the codec which will be used by the codec to read or seek the input stream.
- `codec_decode` is the main decoding API of the codec. This API calls the codec-specific decoding function based on the codec stream type. This API decodes the input raw stream and fills the PCM output samples into codec output PCM buffer. This API gives the information about the number of samples produced by the codec and also gives the pointer of the codec output PCM samples buffer.
- `codec_get_pcm_samples` must be called after the codec's decode API. This API calls the codec specific Get PCM Sample API based on the codec stream type. This API gets the PCM samples from the codec in constant block size and fills them into the output PCM buffer. It returns the number of samples get from the codec and also gives the pointer of the output PCM buffer.
- `codec_reset` calls the codec specific reset API base on stream type and resets the codec.
- `codec_seek` accepts the seek bytes offset converted from the time by application. This API calls the decoder's internal seek API to calculate the actual seek offset which frame boundary aligns. This API returns the actual seek offset.

The basic sequence to use a decoder with the CCI is shown below:



Adding new decoders to the CCI This section explains how to integrate a new decoder in the Common Codec Interface. The CCI assumes the decoder library to be used is in the `\middleware\audio_voice\audiocomponents\decoders\*decoder*\libs\` folder of the maestro framework. The CCI is just a wrapper around a specific implementation. The decoder is expected to be extended as needed to meet the APIs described above.

- Register Decoder Top level APIs in Common Codec Interface
 - Place the decoder lib in libs folder.
 - Add prototypes of the decoder top level APIs in `codec_interface.h` file (located at `maestro\src\cci\inc\` folder).
 - In `codec_interface.c` file (located at `maestro\src\cci\src\`), add top level Decoder APIs in decoder function table.
 - Pseudo code for this is as described below.

```

const codec_interface_function_table_t g_codec_function_table[STREAM_TYPE_COUNT] = {
#ifdef VORBIS_CODEC
{
    &VORBISDecoderGetMemorySize,
    &VORBISDecoderInit,
    &VORBISDecoderDecode,
    NULL,

```

(continues on next page)

(continued from previous page)

```

    NULL,
    &VORBISDecoderSeek,
    &VORBISDecoderGetIOFrameSize,
},
#else
{
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
}
#endif
};

```

- Enable or Disable Decoder
 - Define VORBIS_CODEEC macro in audio_cfg.h file.
 - Comment this macro if you want to disable VORBIS Decoder otherwise keep it defined in order to enable the decoder.
- Add Extract Metadata API for the decoder
 - Add extract metadata API source file for the decoder at streamer/cci/metadata/src/vorbis folder.
 - Add this code in extract metadata lib project space.
 - Build the extract metadata lib and copy that lib to libs folder.
 - Add the desired stream type into ccidec_extract_meta_data API (in codeceextractmeta-data.c file) to call VORBIS Decoder extract metadata API.
- Add stream type of the new decoder in the stream type enum audio_stream_type_t in codec_interface_public_api.h
 - Stream type of the decoder in stream type enum and decoder APIs in decoder function table must be in the same sequence.

Common encoder interface Please see the following section about the [cei](#).

Maestro performance

Memory information The memory usage of the framework components using reference codecs (data obtained from GNU ARM compiler) in bytes is:

text	data	bss	component
48790	2752	4	aac decoder
4348	16400	212	asrc
15512	0	4	flac decoder
76462	16	5013	maestro
34211	0	4	mp3 decoder
211974	0	0	opus
65446	0	4	ssrc
5850	16	12	wav decoder

Maestro framework uses dynamic allocation of audio buffers. The total amount of memory allocated for the pipeline depends on the following parameters:

- Number of elements in the pipeline
- Element types
- Audio stream properties
 - Sampling rate
 - Bit width
 - Channel number
 - Frame size

CPU usage The performance of the pipeline was measured using the real hardware platform (RT1060).

- CPU core clock in MHz: 600.

Pipeline type	Performance MIPS of pipeline (in MHz)
audio source -> audio sink	~10.26 MHz
audio source -> file sink	~9.84 MHz
file source (8-channel PCM) -> audio sink	~16.5 MHz

For performance details about the supported codecs please see audio-voice-components repository documentation.

CEI encoder The Maestro streamer contains an element adapting an extensible set of audio encoders in the form of functions conforming to the CEI (Common Encoder Interface). This element enables the user to choose and configure a suitable encoder at runtime.

Header files CEI itself and the CEI encoders are using following header files, in which you may be interested:

- `cei.h` - contains types used by the element itself and an encoder implementing the CEI
- `cei_enctypes.h` - contains a list of possible encoders and types used for interfacing with a CEI encoder
- `cei_table.h` - contains a table of functions implementing integrated CEI encoders

Instantiating the element This element's index is `ELEMENT_ENCODER_INDEX` and its type is `TYPE_ELEMENT_ENCODER`, as defined in `streamer_api.h`. It has one source pad (data input) and one sink pad (data output). It is initialized like any other element, meaning that it is instantiated and inserted into the pipeline using the `create_element`, `add_element_pipeline` and `link_elements` functions. Inversely, for destroying the element, the `unlink_elements`, `remove_element_pipeline` and `destroy_element` are used. This element alone does not depend on any additional software layers other than these required by the Maestro streamer itself, so no pre-initialization before this element instantiation is necessary.

Element properties Use Maestro streamer property API (`streamer_set_property` and `streamer_get_property`) for setting or getting these. The constants are defined in `streamer_element_properties.h`.

- `PROP_ENCODER_CHUNK_SIZE`

- **Synopsis:** Determines the length of a chunk pulled from the sibling of the source pad and essentially influences the size of allocated buffers. If the actual amount of data pulled is smaller, the rest is zero-filled.
- **Type:** unsigned 32-bit integer
- **Default value:** 1920
- **Constraints:**
 - * Must be bigger than zero, otherwise `STREAM_ERR_INVALID_ARGS` is returned.
 - * Cannot be changed if the actual encoder has been created. If done so, `STREAM_ERR_ELEMENT_BAD_STATUS` is returned.
- `PROP_ENCODER_TYPE`
 - **Synopsis:** Determines the exact encoder (CEI implementation) to be used.
 - **Type:** `CeiEncoderType` (`cei_enctypes.h`)
 - **Default value:** `CEIENC_LAST`
 - **Constraints:**
 - * Must not be equal to `CEIENC_LAST`, otherwise `STREAM_ERR_INVALID_ARGS` will be returned.
 - * Selected encoder must be implemented, otherwise `STREAM_ERR_INVALID_ARGS` will be returned.
 - * Cannot be changed if the actual encoder has been created. If done so, `STREAM_ERR_ELEMENT_BAD_STATUS` will be returned.
 - **Behaviour influenced:** The encoder element process function will return `FLOW_ERROR` if this property isn't set.
- `PROP_ENCODER_CONFIG`
 - **Synopsis:** Determines encoder-specific configuration (application, bitrate, ...).
 - **Type:** Pointer to the encoder-specific configuration structure.
 - **Default value:** Determined by the encoder.
 - **Constraints:**
 - * The encoder has to be configurable. If it is not, `STREAM_ERR_ERR_GENERAL` will be returned on any access.
 - * The structure has to conform to the encoder requirements. If the encoder returns an error code, `STREAM_ERR_GENERAL` will be returned.
- `PROP_ENCODER_BITSTREAMINFO`
 - **Synopsis:** Specifies information about the incoming bitstream (sample rate, sample depth, ...).
 - **Type:** Pointer to `CeiBitstreamInfo` (`cei_enctypes.h`).
 - **Default value:**

```
(CeiBitstreamInfo) {
    .sample_rate = 0,
    .num_channels = 0,
    .endian = AF_LITTLE_ENDIAN,
    .sign = TRUE,
    .sample_size = 0,
    .interleaved = TRUE
}
```

– **Constraints:**

- * Cannot be changed if the actual encoder has been created. If done so, `STREAM_ERR_ELEMENT_BAD_STATUS` will be returned.
- * As of now, only bitstreams containing 16-bit interleaved (if 2 or more channels will be encoded) samples are supported. If anything else was set to the `sample_size` and `interleaved_members`, `STREAM_ERR_INVALID_ARGS` will be returned.

– **Behaviour influenced:**

- * Given the characteristics of some elements available, different packets of data (header and payload, referred to as “chunk” above) may be pulled by this element. Each packet can contain a different header, which may or may not contain useful information about the bitstream. If a packet with the `AudioPacketHeader` (`todofile.h`) is pulled at first and any other iteration of the streamer pipeline, the bitstream parameters configured by this property are implicitly available and are not expected to be specified by the user. Other packet header types (such as `RawPacketHeader`) don’t contain any bitstream parameters and require the user to specify the parameters manually using this property. Failure to do so will result in the element’s process function returning `FLOW_ERROR`. Same situation will occur if a packet with the `AudioPacketHeader` is received and its contents differ from the already acquired bitstream parameters.
- * As of now, CEI is defined to work with 16-bit signed little-endian (s16le) samples, which are interleaved if the bitstream contains more than one channels. This element handles endianness and unsigned to signed conversion.

CEI definition - implementing your own encoder The CEI defines following function pointer types:

- `CeiFnGetMemorySize`: Returns number of bytes required for encoder state for a given number of channels.
- `CeiFnEncoderInit`: Initialize an encoder for a given sample rate and channel count.
- `CeiFnEncoderGetConfig`: Copy current or default configuration to a given structure pointer.
- `CeiFnEncoderSetConfig`: Configure the encoder from a given structure pointer.
- `CeiFnEncode`: Encode a given buffer to a given output buffer.

Detailed descriptions of function behaviour, parameters and expected return values are available as docblocks in the `cei.h` file.

Each encoder is implemented as a set of pointers pointing to functions conforming to these types, grouped in the `CeiEncoderFunctions` structure. Specifying the `CeiEncoderGetConfig` `fnGetConfig` and `CeiFnEncoderSetConfig` `fnSetConfig` members is optional, as an encoder does not have to be configurable. If so desired, specify `NULL`. Implementation of the remaining functions is mandatory, however. If at least one of these functions isn’t implemented and `NULL` is specified instead, the encoder will be considered as not implemented.

To register an implemented encoder with the element, add a new entry to the `CeiEncoderType` enum and add the `CeiEncoderFunctions` struct value to the table `CeiEncoderFunctions` `ceiEncTable[]` located in the `cei_table.h` header file. Note and match the order of items in that table, as a `CeiEncoderType` value is used as an index. Same goes for the `size_t` `ceiEncConfigSizeTable[]`. If configuration is not applicable, specify 0 at the appropriate index. If configuration is applicable, describe the configuration structure in the `cei_encetypes.h` header file and add its size to that table.

Maestro playback example

Table of content

- [Overview](#)
- [Hardware requirements](#)
- [Hardware modifications](#)
- [Preparation](#)
- [Running the demo](#)
- [Example configuration](#)
- [Functionality](#)
- [States](#)
- [Commands in detail](#)
- [Processing Time](#)

Overview The Maestro playback example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console and the audio files are read from the SD card.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback of encoded files from an SD card with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-channel** - The mode demonstrates playback of raw PCM files from an SD card with 2 or 8 channels, 96kHz sample rate and 32 bit width. The decoders and synchronous sample rate converter are not supported in this mode. The Multi-channel mode is only supported on selected platforms, see the table below. The [Example configuration](#) section contains information on how to enable it.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and [Hardware modifications](#) or [Example configuration](#) sections before running the demo.

Limitations:

- Note:
 - *LPCXpresso55s69* - MCUXpresso IDE project default debug console is semihost
- Decoder:
 - **AAC:**
 - * The reference decoder is supported only in the MCUXpresso IDE and ARMGCC.
 - **FLAC:**
 - * *LPCXpresso55s69* - When playing FLAC audio files with too small frame size (block size), the audio output may be distorted because the board is not fast enough.
 - **OPUS:**
 - * *LPCXpresso55s69* - The decoder is disabled due to insufficient memory may be distorted because the board is not fast enough.
- Sample rate converter:
 - **SSRC:**

- * *LPCXpresso55s69* - When a memory allocation ERROR occurs, it is necessary to disable the SSRC element due to insufficient memory.

Known issues:

- Decoder:
 - **MP3:**
 - * The reference decoder has issues with some of the files. One of the channels can be sometimes distorted or missing parts of the signal.
 - **OPUS:**
 - * The decoder doesn't support all the combinations of frame sizes and sample rates. The application might crash when playing an unsupported file.

More information about supported features can be found on the [Supported features](#) page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- SD card with supported audio files
- Personal computer
- Optional:
 - Audio expansion board [AUD-EXP-42448 \(REV B\)](#)

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
 1. Please remove below resistors if on board wifi chip is not DNP:
 - R228, R229, R232, R234
 2. Please make sure R136 is weld for GPIO card detect.

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board.
2. Open a serial terminal with the following settings:
 - 115200 baud rate
 - 8 data bits
 - No parity
 - One stop bit
 - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.

5. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
*****
Maestro audio playback demo start
*****

[APP_Main_Task] started

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type `help` to see the command list. Similar description will be displayed on serial console (*If multi-channel playback mode is enabled, the description is slightly different*):

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"file": Perform audio file decode and playback

USAGE: file [stop|pause|volume|seek|play|list|info]
stop           Stops actual playback.
pause          Pause actual track or resume if already paused.
volume <volume> Set volume. The volume can be set from 0 to 100.
seek <seek_time> Seek currently paused track. Seek time is absolute time in milliseconds.
play <filename> Select audio track to play.
list           List audio files available on mounted SD card.
info           Prints playback info.
```

Details of commands can be found [here](#).

Example configuration The example can be configured by user. Before configuration, please check the [table](#) to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**

- Add the MULTICHANNEL_EXAMPLE symbol to preprocessor defines on project level.
- Connect AUD-EXP-42448 (see the point below).

- **Connect AUD-EXP-42448:**

- *EVKC-MIMXRT1060*:
 1. Disconnect the power supply for safety reasons.
 2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
 3. Uninstall J99.
 4. Set the DEMO_CODEC_WM8962 macro to 0 in the `app_definitions.h` file

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:#ff9900
```

```
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:#ff999c
```

```
B((Idle)):::state --> C{Duration
== 0?}:::condition
```

```

C -- No --> E{Duration
< 0?}:::condition
C -- Yes --> D[Error: Incorrect
command parameter]:::error
D --> B
E -- Yes --> G[playing]:::function
G --> H((Running)):::state
H --> H
E -- No --> F[playing]:::function
F --> I((Running)):::state
I --> J{Duration
expired?}:::condition
J -- No --> I
J -- Yes --> B

```

Supported features The current version of the audio framework supports several optional features. These can be limited to some MCU cores or development boards variants. More information about support can be found on the specific example page:

- [maestro_playback](#)
- [maestro_record](#)
- [maestro_usb_mic](#)
- [maestro_usb_speaker](#)

Some features are delivered as prebuilt library and the binaries can be found in the `\middleware\audio_voice\components\*component*\libs` folder. The source code of some features can be found in the `\middleware\audio_voice\maestro\src` folder.

Decoders Supported decoders and its options are:

Decoder	Sample rates [kHz]	Number of channels	Bit depth
AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
FLAC	8, 11.025, 12, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	16
MP3	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
OPUS	8, 16, 24, 48	1, 2 (mono/stereo)	16
WAV	8, 11.025, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	8, 16, 24

For more details about the reference decoders please see audio-voice-components repository documentation `\middleware\audio_voice\components\`.

Encoders

- **OPUS encoder** - The current version of the audio framework only supports a OPUS encoder. For more details about the encoder please see the following [link](#).

Sample rate converters

- **SSRC** - Synchronous sample rate converter. More details about SSRC are available in the User Guide, which is located in `middleware\audio_voice\components\ssrc\doc\`.
- **ASRC** - Asynchronous sample rate converter is not used in our examples, but it is part of the maestro middleware and can be enabled. To enable ASRC, the `maestro_framework_asrc` and `CMSIS_DSP_Library_Source` components must be added to the project. Furthermore, it is necessary to switch from Redlib to Newlib (semihost) library and add a platform definition

to the project (e.g. for RT1170: PLATFORM_RT1170_CORTEXM7). Supported platforms can be found in the `PL_platformTypes.h` file. More details about ASRC are available in the User Guide, which is located in `middleware\audio_voice\components\asrc\doc\`.

Additional libraries

- **VIT** - Voice Intelligent Technology (VIT) Wake Word and Voice Command Engines provide free, ready to use voice UI enablement for developers. It enables customer-defined wake words and commands using free online tools. More details about VIT are available in the VIT package, which is located in `middleware\audio_voice\components\vit\{platform}\Doc\` (depending on the platform) or via following [link](#).

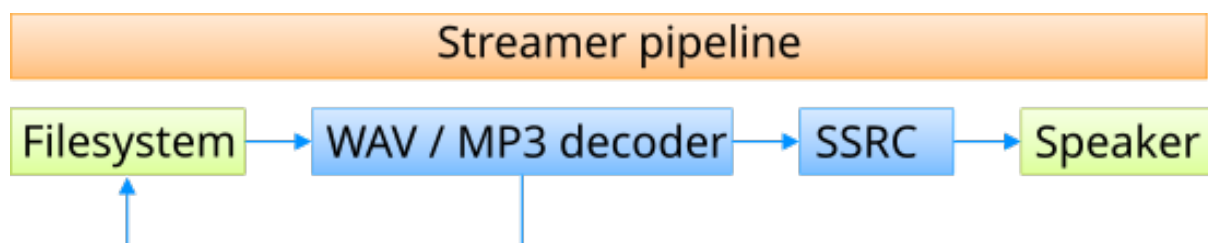
Processing Time

Table of content

- [Maestro playback example](#)
- [Maestro record example](#)

The individual time measurements were conducted using a logic analyzer by monitoring changes in the GPIO port levels on the EVKC-MIMXRT1060 development board. These measurements were executed for each individual pipeline run, capturing the timing at each corresponding element, and, when relevant, the interconnections between these elements.

Maestro playback example For the Maestro playback example the following reference audio file was used: `test_48khz_16bit_2ch.wav`. In this example, the pipeline depicted in the diagram was considered. Media codecs WAV and MP3 were taken into account. To compare the times spent on the SSRC block, sampling rates for both codecs were selected: 44.1 kHz and 48 kHz.



The measurement of streamer pipeline run started at the beginning of `streamer_process_pipelines()` in `streamer.c` and ended in the function `streamer_pcm_write()` in `streamer_pcm.c` just before the output buffer.

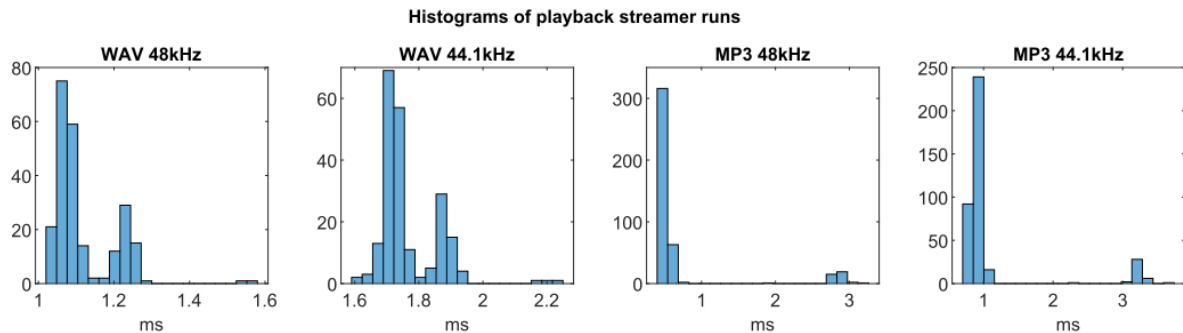
In the scenario involving the WAV codec, the audio file was accessed in every iteration of the streamer pipeline. Meaning, during each run, the file was read directly from the SD card. However, in the case of the MP3 codec, where data processing necessitates complete MP3 frames, the file wasn't read during every run. Rather, it was accessed periodically, triggered when the codec buffer lacked a complete MP3 frame of data. The total time spent on codec processing varies significantly depending on the type and implementation of the codec. For certain types of codecs, like FLAC, there may be multiple file accesses during a single pipeline run. The provided values are specific to the reference implementation. For details about the codecs please see `audio-voice-components` documentation `middleware\audio_voice\components\`.

The duration of the streamer pipeline illustrates that with a sampling frequency of 48 kHz, there is no resampling occurring at the SSRC element. Consequently, the overall pipeline time is lower than in the case of 44.1 kHz audio, where resampling takes place.

To enhance comprehension of the system's behavior, histograms of the pipeline run times and its elements are included. The greater time variance with the MP3 codec is precisely due to

the absence of file reads in every run. In clusters with shorter times, there are no file accesses, while in clusters with longer times, file reads occur. This indicates that the majority of runs do not involve file access.

	WAV 48 kHz	WAV 44 kHz	MP3 48 kHz file read	MP3 48 kHz w/o file read	MP3 44 kHz file read	MP3 44 kHz w/o file read
mear	1.11 ms	1.76 ms	2.87 ms	0.51 ms	3.22 ms	0.89 ms
min	1.03 ms	1.60 ms	2.74 ms	0.41 ms	2.33 ms	0.74 ms
max	1.29 ms	2.23 ms	3.24 ms	1.83 ms	3.73 ms	1.12 ms



Time on each element In the tables and histograms below, the timings for individual elements and their connections are provided. Given that the file reading function was invoked during the codec's operation, the tables for individual elements display the total time on the codec element, the time on the codec element before the file read, and the time on the codec element after the file read. The individual blocks in the tables are as follows:

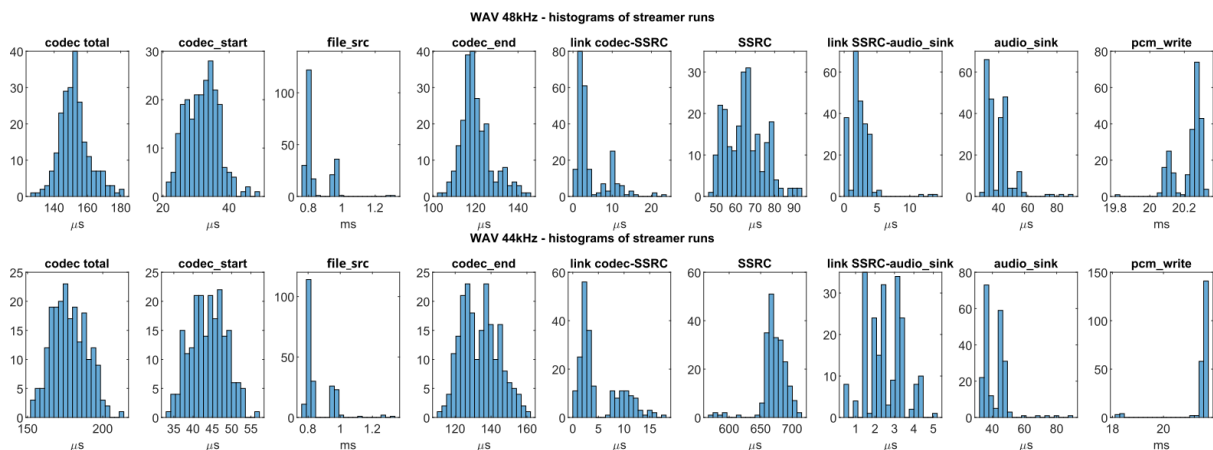
- **streamer** - total time of one pipeline run without time on output buffers
- **codec start** - time on decoder before file read
- **codec end** - time on decoder after file read
- **codec total** - codec_start+codec_end
- **file_src** - file reading time
- **SSRC_proc** - time on SSRC element
- **audio_sink** - time on audio sink without output buffers
- **pcm_write** - time on output buffers
- **link** - time on element links

The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **codec** - decoder_sink_pad_process_handler():decoder_pads.c
- **file_src** - filesrc_read():file_src_rtos.c
- **SSRC_proc** - SSRC_Proc_Execute():ssrc_proc.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

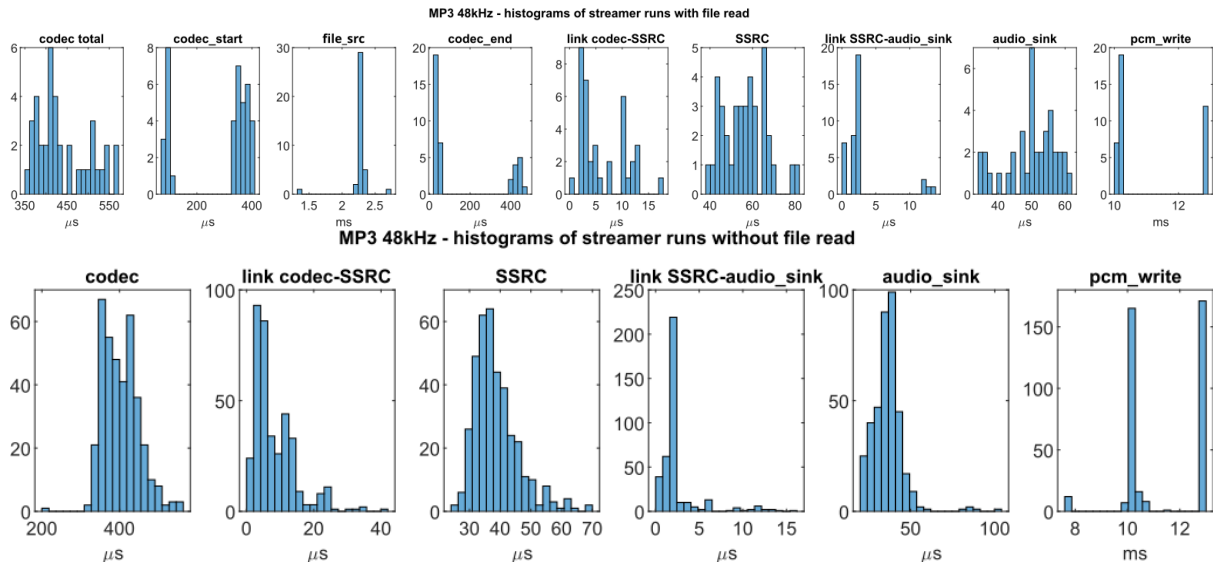
WAV 48kHz	stream	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_	link audio_sink	SSRC- audio_sin	audio_sin	pcm_write
mean	1.119 ms	152 μ s	31 μ s	0.843 ms	120 μ s	5 μ s	64 μ s	2 μ s		40 μ s	20.228 ms
min	1.026 ms	125 μ s	21 μ s	0.773 ms	104 μ s	<1 μ s	47 μ s	<1 μ s		30 μ s	19.805 ms
max	1.290 ms	193 μ s	49 μ s	1.311 ms	144 μ s	23 μ s	93 μ s	14 μ s		91 μ s	20.324 ms

WAV 44kHz	stream	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_	link audio_sink	SSRC- audio_sin	audio_sin	pcm_write
mean	1.765 ms	178 μ s	44 μ s	0.853 ms	134 μ s	5 μ s	671 μ s	3 μ s		42 μ s	21.472 ms
min	1.604 ms	145 μ s	33 μ s	0.770 ms	112 μ s	<1 μ s	574 μ s	<1 μ s		33 μ s	18.163 ms
max	2.233 ms	218 μ s	57 μ s	1.335 ms	161 μ s	18 μ s	715 μ s	5 μ s		89 μ s	21.746 ms



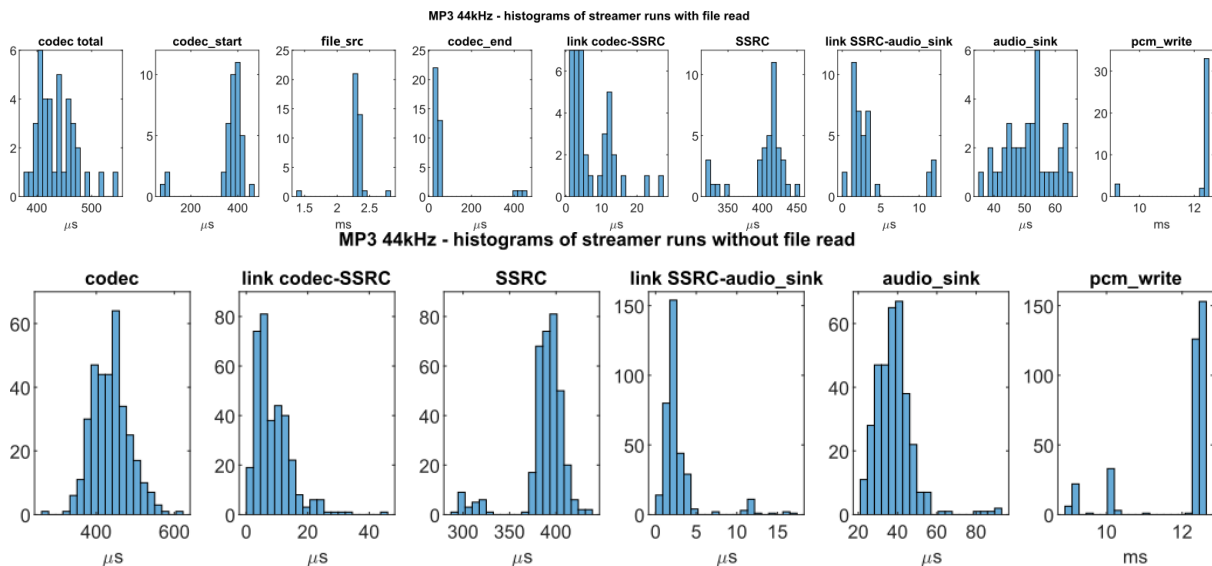
MP3 48 kHz w/ file read	stream	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_	link SSRC- audio_sink	au- dio_sir	pcm_write
mean	2.871 ms	441 µs	279 µs	2.271 ms	162 µs	6 µs	56 µs	3 µs	50 µs	11.019 ms
min	2.739 ms	353 µs	74 µs	1.353 ms	26 µs	<1 µs	40 µs	<1 µs	34 µs	10.091 ms
max	3.244 ms	570 µs	409 µs	2.728 ms	467 µs	18 µs	80 µs	14 µs	62 µs	12.910 ms

MP3 kHz w/o file read	48	strear	codec total	codec start	file_s	codec end	link codec- SSRC	SSRC_l	link SSRC- audio_sink	au- dio_sir	pcm_write
mean		0.508 ms	403 μs	x	x	x	8 μs	39 μs	3 μs	36 μs	11.326 ms
min		0.407 ms	208 μs	x	x	x	<1 μs	25 μs	<1 μs	21 μs	7.715 ms
max		1.834 ms	563 μs	x	x	x	41 μs	69 μs	16 μs	104 μs	12.941 ms

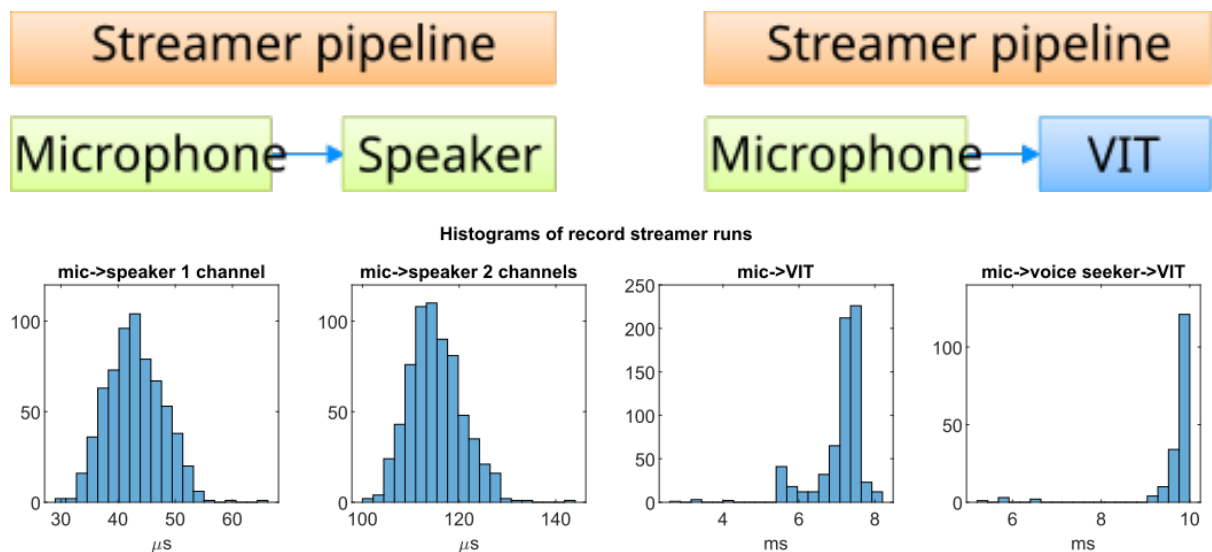


MP3 44 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec- SSRC	SSRC_l	link SSRC- audio_sink	au- dio_sir	pcm_write
mean	3.217 ms	436 μs	367 μs	2.300 ms	66 μs	7 μs	403 μs	3 μs	51 μs	12.188 ms
min	2.329 ms	383 μs	73 μs	1.411 ms	26 μs	2 μs	318 μs	<1 μs	35 μs	9.119 ms
max	3.726 ms	547 μs	464 μs	2.801 ms	441 μs	27 μs	454 μs	12 μs	65 μs	12.529 ms

MP3 kHz w/o file read	44	strear	codec total	codec start	file_s	codec end	link codec- SSRC	SSRC_l	link SSRC- audio_sink	au- dio_sir	pcm_write
mean		0.891 ms	437 μs	x	x	x	9 μs	388 μs	3 μs	38 μs	11.934 ms
min		0.738 ms	268 μs	x	x	x	<1 μs	290 μs	<1 μs	22 μs	8.964 ms
max		1.115 ms	620 μs	x	x	x	45 μs	438 μs	17 μs	92 μs	12.624 ms



Maestro record example Typical execution times of the streamer pipeline and its individual elements for the EVKC-MIMXRT1060 development board are detailed in the following tables. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines are depicted in the figure below. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone and VIT blocks. The measurement of streamer pipeline run started at the beginning of `streamer_process_pipelines();streamer.c` and ended in the function `streamer_pcm_write(): streamer_pcm.c` just before the output buffer.



The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers and without time reading from the microphone
- **audio_src_start** - time on audio src before reading from the microphone
- **audio_src_end** - time on audio src after reading from the microphone
- **pcm_read** - reading from the microphone
- **vit** - time on VIT element
- **audio_sink** - time on audio sink without output buffers

- **pcm_write** - time on output buffers
- **link** - time on element links

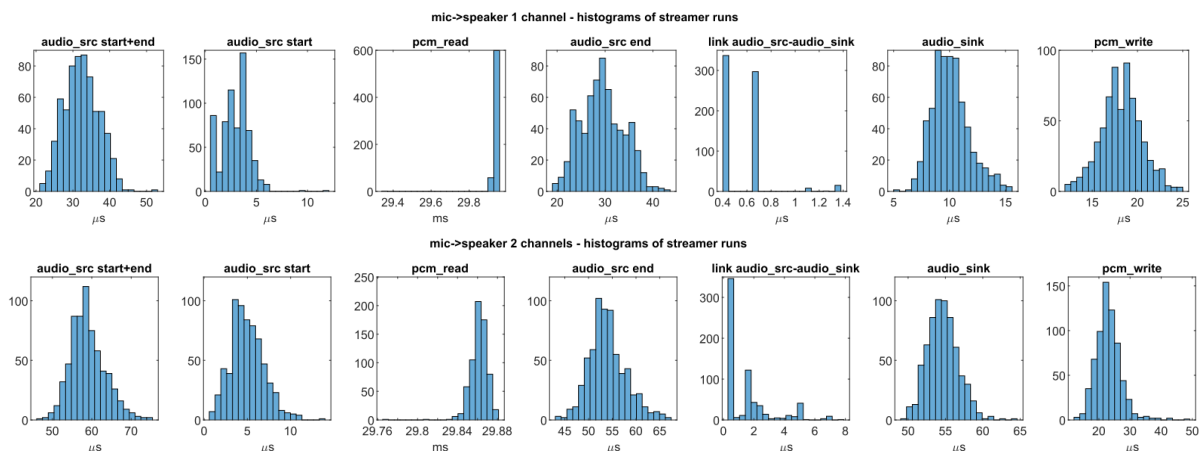
The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **audio_src** - audiosrc_src_process():audio_src.c
- **pcm_read** - streamer_pcm_read():streamer_pcm.c
- **vit** - vitsink_sink_pad_chain_handler():vit_sink.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

Pipeline Microphone -> Speaker

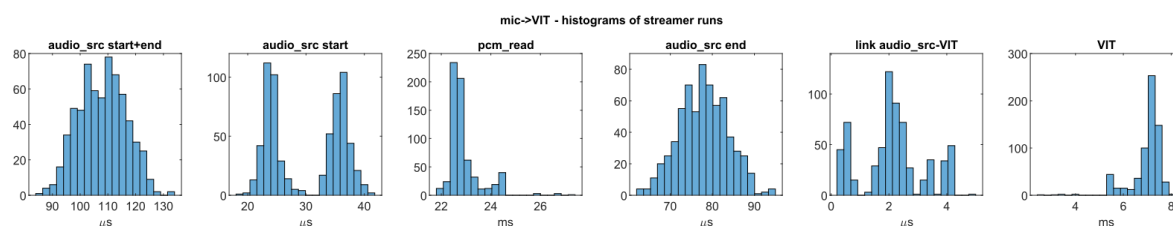
microphone speaker mono	->	stream	audio_src_start	pcm_read	audio_src_end	link audio_src-audio_sink	audio_sink	pcm_write
mean		43 μ s	3 μ s	29.938 ms	29 μ s	<1 μ s	10 μ s	18 μ s
min		26 μ s	<1 μ s	29.350 ms	19 μ s	<1 μ s	5 μ s	12 μ s
max		72 μ s	12 μ s	29.957 ms	44 μ s	1 μ s	15 μ s	25 μ s

microphone speaker stereo	->	stream	audio_src_start	pcm_read	audio_src_end	link audio_src-audio_sink	audio_sink	pcm_write
mean		115 μ s	5 μ s	29.861 ms	54 μ s	2 μ s	55 μ s	23 μ s
min		94 μ s	<1 μ s	29.768 ms	43 μ s	<1 μ s	50 μ s	12 μ s
max		154 μ s	14 μ s	29.880 ms	67 μ s	8 μ s	65 μ s	49 μ s



Pipeline Microphone -> VIT

microphone -> VIT	streamer	audio_src_start	pcm_read	audio_src_end	link audio_src-vit	vit
mean	7.380 ms	30 μ s	22.624 ms	78 μ s	2 μ s	7.261 ms
min	2.641 ms	10 μ s	2.2265 ms	58 μ s	<1 μ s	2.559 ms
max	7.780 ms	42 μ s	2.7341 ms	94 μ s	5 μ s	7.624 ms



Maestro on Zephyr

- Based on and tested with Zephyr version, given by tag v4.0.0
- Tested with Zephyr SDK version 16.4
- To see the pre-built documentation, see: [README.html](#). Also see the [documentation section](#).

Maestro sample for recording data from microphone to RAM

Description This sample records data from microphone (alias dmich0 in devicetree) and stores them to a buffer in RAM.

Currently one PDM channel with fixed 16 kHz sample rate and 16 bit sample width is supported. For configuration options, see Kconfig and prj.conf.

User Input/Output

- Input:
 - None.
- Output:
 - UART Output:
 - Demo result: OK if everything went OK
 - Demo result: FAIL otherwise

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro voice detection sample using VIT

Description Records data from microphone (alias `dmic0` in devicetree) and detects voice commands from selected language model. Detected commands are printed via UART.

Language model may be changed via Kconfig using `CONFIG_MAESTRO_EXAMPLE_VIT_LANGUAGE` selection. For other configuration options, see example's Kconfig and `prj.conf`.

This project requires an NXP board supported by the VIT library.

The example has to be modified if a new board needs to be added. Please create an issue in that case.

User Input/Output

- Input:
 - None.
- Output:
 - UART Output:
 - List of voice commands the model can detect (printed immediately after start)
 - `<Specific voice command>` if voice command was detected
 - Demo result: FAIL otherwise

Dependencies

- VIT library: <https://www.nxp.com/design/design-center/software/embedded-software/voice-intelligent-technology-wake-word-and-voice-command-engines:VOICE-INTELLIGENT-TECHNOLOGY>

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro decoder sample

Description Tests and demonstrates decoder functionality in Maestro pipeline.

Supported decoders:

- MP3
- WAV
- AAC
- FLAC
- OPUS with OGG envelop
- (RAW OPUS - TBD)

Data Input:

- Prepared encoded audio data (part of Maestro repository, folder `zephyr/audioTracks`)
- Prepared decoded audio data (RAW PCM format, part of Maestro repository, folder `zephyr/audioTracks`)

Function:

1. Loads encoded data into source buffer stored in RAM

2. Decodes audio data using selected decoder and stores data in RAM
3. Compares prepared data with decoded data to check if its the same
4. Prints Demo result: OK or Demo result: FAIL via UART

User Input/Output

- Input:
None
- Output:
UART Output
 - Demo result: OK if everything went OK
 - Demo result: FAIL otherwise

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Decoder libraries

Configuration

- See prj.conf for user input sections
 - Selecting decoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_DECODER_SELECTED in prj.conf file. When no decoder is selected, default one (WAV) is used instead.
 - System settings should be modified (stack size, heap size) based on selected decoder and system capabilities/requirements in prj.conf.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms

 Currently tested for:

- RD_RW612_BGA - Working decoders: FLAC, WAV, OPUS OGG

Maestro encoder sample

Description Tests and demonstrates encoder functionality in Maestro pipeline.

Supported encoders:

- OPUS with OGG envelop - TBD
- RAW OPUS - TBD

Input:

- Prepared decoded audio data (RAW PCM format, part of Maestro repository)
- Prepared encoded audio data (part of Maestro repository)

Function:

1. Loads RAW data into source buffer stored in RAM
2. Encodes audio data using selected encoder and stores data in RAM
3. Compares prepared data with decoded data if same
4. Prints Demo result: OK or Demo result: FAIL via UART

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Encoder libraries

User Input/Output

Input:

- None

Output:

- UART Output
 - Demo result: OK if everything went OK
 - Demo result: FAIL otherwise

Configuration

- See prj.conf for user input sections
 - Selecting encoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_ENCODER_SELECTED in prj.conf file. When no encoder is selected, default one (OPUS) is used instead.
 - System settings should be modified (stack size, heap size) based on selected encoder and system capabilities/requirements in prj.conf file.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms

Currently tested for:

- RD_RW612_BGA - Working encoders: None.

Maestro mem2mem sample

Description Tests basic memory to memory pipeline.

Function:

1. Moves generated data with fixed size of 256B from memory source to memory sink.
2. Compares copied data to check if they're the same.
3. Returns Demo result: OK or Demo result: FAIL via UART.

- [Maestro environment setup](#)
- [Build and run Maestro example](#)
 - [Using command line](#)
 - [Using MCUXpresso for VS Code](#)
- [Folder structure](#)
- [Supported elements and libraries](#)
- [Examples support](#)
- [Creating your own example](#)
- [Documentation](#)
- [FAQ](#)

Maestro environment setup Follow these steps to set up a Maestro development environment on your machine.

1. If you haven't already, please follow [this guide](#) to set up a Zephyr development environment and its dependencies first:

- Cmake
- Python
- Devicetree compiler
- West
- Zephyr SDK bundle

2. Get Maestro. You can pick either of the options listed below. If you need help deciding which option is the best fit for your needs, please see the [FAQ](#).

- Freestanding Maestro - This option pulls in only Maestro's necessary dependencies.

Run:

```
1. west init -m <maestro repository url> --mr <revision> --mf west-freestanding.yml  
   ↳<foldername>  
2. cd <foldername>  
3. west update
```

- Maestro as a Zephyr module

To include Maestro into Zephyr, update Zephyr's west.yml file:

```
projects:  
  name: maestro  
  url: <maestro repository url>  
  revision: <revision with Zephyr support>  
  path: modules/audio/maestro  
  import: west.yml
```

Then run west update maestro command.

Build and run Maestro example These steps will guide you through building and running Maestro samples. You can use either the command line utilizing Zephyr's powerful west tool or you can use VS Code's GUI. Detailed steps for both options are listed below.

Using command line See Zephyr's [Building, Flashing and Debugging](#) guide if you aren't familiar with it yet.

1. To **build** a project, run:

```
west build -b <board> -d <output build directory> <path to example> -p
```

For example, this compiles VIT example for rd_rw612_bga board:

```
1. cd maestro/zephyr  
2. west build -b rd_rw612_bga -d build samples/vit -p
```

2. To **run** a project, run:

```
west flash -d <directory>
```

e.g.:

```
west flash -d build
```

3. To **debug** a project, run:

```
west debug -d <directory>
```

e.g.:

```
west debug -d build
```

Using MCUXpresso for VS Code For this you have to have NXP's **MCUXpresso for VS Code extension** installed.

1. Import your topdir as a repository to MCUXpresso for VS Code:
 - Open the MCUXpresso Extension. In the *Quickstart Panel* click *Import Repository*.
 - In the displayed menu click *LOCAL* tab and select the folder location of your *topdir*.
 - Click *Import*.
 - The repository is successfully added to the *Installed Repositories* view once the import is successful.
2. To import any project from the imported repository:
 - In the *Quickstart Panel* click *Import Example from Repository*.
 - For **Repository** select *your imported repository*.
 - For **Zephyr SDK** the installed Zephyr SDK is selected automatically. If not, select one.
 - For **Board** select your board (*make sure you've selected the correct revision*).
 - For **Template** select the folder path to your project.
 - Click the *Create* button.
3. Build the project by clicking the *Build Selected* icon (displayed on hover) in the extension's *Projects* view. After the build, the debug console window displays the memory usage (or compiler errors if any).
4. Debug the project by clicking the *Debug* (play) icon (displayed on hover) in the extension's *Projects* view.
5. The execution will pause. To continue execution click *Continue* on the debug options.
6. In the *SERIAL MONITOR* tab of your console panel, the application prints the Zephyr boot banner during startup and then prints the test results.

Folder structure

```
maestro/
...
zephyr/      All Zephyr related files
  samples/   Sample examples
  tests/     Tests
  audioTracks/ Audio tracks for testing
  doc/       Documentation configuration for Sphinx
  wrappers/  NXP SDK Wrappers
  scripts/   Helper scripts, mostly for testing
  module.yml Defines module name, Cmake and Kconfig locations
  CMakeList.txt Defines module's build process
  Kconfig    Defines module's configuration
  osa/       Deprecated. OSA port for Zephyr
...
```

Supported elements and libraries Here is the list of all features currently supported in Maestro on Zephyr. Our goal is to support all features in Maestro on Zephyr that are already supported in Maestro on NXP's SDK and to extend them further.

Supported elements:

- Memory source
- Memory sink
- Audio source
- Audio sink
- Process sink
- Decoder
- Encoder

Supported decoders:

- WAV
- MP3
- FLAC
- OPUS OGG
- AAC

Supported encoders:

- OPUS RAW

Supported libraries:

- [VIT](#)

Examples support All included examples use UART as output. Examples are located in `zephyr/tests` and `zephyr/samples` directories.

List of included examples:

- [Maestro sample for recording data from microphone to RAM](#)
- [Maestro voice detection sample using VIT](#)
- [Maestro encoder sample](#)
- [Maestro decoder sample](#)
- [Maestro mem2mem sample](#)

Examples support for specific boards:

Example	RDRW612BGA	LPCx-presso55s69	MIMXRT1060EVKE	MIMXRT1170EVKB
Record	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
VIT	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
Encoder	In progress: OPUS RAW	TO BE TESTED	TO BE TESTED	TO BE TESTED
Decoder	YES - WAV, FLAC, OPUS OGG	TO BE TESTED	TO BE TESTED	TO BE TESTED
Mem2mem	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED

Creating your own example There are two ways to create your own example - you can either one of the included examples as a reference or you can create your own example from scratch by hand.

When creating your own example from scratch, set `CONFIG_MAESTRO_AUDIO_FRAMEWORK=y` in your `prj.conf` file. Then you can start enabling specific elements by setting `CONFIG_MAESTRO_ELEMENT_<NAME>_ENABLE=y`.

However, the recommended way to edit config options is to open `gui-config` (or `menuconfig`) by calling `west build -t guiconfig`. Then you can use the graphical interface to interactively turn on/off the features you need.

Documentation Please note, Maestro documentation is under reconstruction. It is currently mixing several tools and formats.

To see the pre-generated Maestro Zephyr documentation, see `zephyr/doc/doc/README.html`

To generate the Zephyr documentation, go under `zephyr/doc` folder and execute `make html`. Sphinx version `sphinx-build 8.1.3` must be installed. Open `doc/doc/html/README.html` afterwards.

To see Maestro core documentation, go to the Maestro top directory and see `README.md`.

FAQ

1. Should I choose the freestanding version of Maestro or should integrate it into my west instead?
 - Freestanding version of Maestro pulls in all the dependencies it needs including Zephyr itself.
 - Integrating it as a module is easier if you already have your Zephyr environment set up.

Maestro Audio Framework changelog

2.0.2

- Removed VoiceSeeker support

2.0.1

- Fixed filesrc buffer alignment

2.0.0 (newest)

- Added Zephyr port, see [Zephyr README](#).
 - Possible to use standalone version, pulling its own Zephyr and dependencies
 - Possible to import it as a module in your Zephyr project
- Changed build system - newly uses Kconfig and Cmake
- Supports NXP MCUXSDK (previously 2.x)
- Changed folder structure and names to improve readability (description may be found in [README](#))
- Removed audio libraries and placed into audio-voice-components repository
- Added libraries are pulled into the build via Kconfig and Cmake

- Changed Maestro library core - minor changes

1.8.0

- New platforms support: MCX-N5XX-EVK, FRDMMCXN236 and RD-RW612-BGA
- Fixed compilation warnings
- Documentation improvements and updates
 - Added section with processing time information
 - Added application state diagrams
- Various updates and fixes

1.7.0

- Removed EAP support for future SDK releases
- Created new API for audio_sink and audio_src to support USB source, sink
- ASRC library integrated
- License changed to BSD 3-Clause
- Improved pipeline creation API
- Fixed compilation warnings in Opus
- Various other improvements and bug fixes

1.6.0

- Up to 2 parallel pipelines supported
- Synchronous Sample Rate Converter support Added
- Various improvements and bug fixes

1.5.0

- Enabled switching from 2 to 4 channel output during processing
- PadReturn type has been replaced by FlowReturn
- Support of AAC, WAV, FLAC decoders
- Renamed eap element to audio_proc element
- Added audio_proc to VIT pipeline to support VoiceSeeker
- Minor bug fixes

1.4.0

- Use Opusfile lib for Ogg Opus decoder
- Refactor code, fix issues found in unit tests
- Various bug fixes

1.3.0

- Make Maestro framework open source (except mp3 and wav decoder)
- Refactor code, remove unused parts, add comments

1.2.0

- Unified buffering in audio source, audio sink
- Various improvements and bug fixes

1.0_rev0

- Initial version of framework with support for Cortex-M7 platforms

Chapter 4

RTOS

4.1 FreeRTOS

4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

[FreeRTOS kernel for MCUXpresso SDK Readme](#)

[FreeRTOS kernel for MCUXpresso SDK ChangeLog](#)

[FreeRTOS kernel Readme](#)

4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

[Readme](#)

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

4.1.5 corejson

JSON parser.

Readme

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

4.1.7 corepkcs11

PKCS #11 key management library.

Readme

4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme