



MCUXpresso SDK Documentation

Release 26.03.00



NXP
Mar 24, 2026



Table of contents

1	FRDM-MCXC162	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	59
1.3.1	Getting Started with MCUXpresso SDK Repository	59
1.4	Release Notes	66
1.4.1	MCUXpresso SDK Release Notes	66
1.5	ChangeLog	69
1.5.1	MCUXpresso SDK Changelog	69
1.6	Driver API Reference Manual	135
1.7	Middleware Documentation	135
1.7.1	MCU Boot	135
1.7.2	FreeMASTER	135
1.7.3	FreeRTOS	135
1.7.4	File systemFatfs	135
2	MCXC162	137
2.1	Clock Driver	137
2.2	CRC: Cyclic Redundancy Check Driver	159
2.3	CTIMER: Standard counter/timers	161
2.4	eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	171
2.5	eDMA core Driver	202
2.6	eDMA soc Driver	209
2.7	EIM: error injection module	209
2.8	EQDC: Enhanced Quadrature Encoder/Decoder	209
2.9	ERM: error recording module	227
2.10	FGPIO Driver	229
2.11	C40ESP3 Flash Driver	229
2.12	FREQME: Frequency Measurement	232
2.13	GPIO: General-Purpose Input/Output Driver	232
2.14	GPIO Driver	235
2.15	INPUTMUX: Input Multiplexing Driver	241
2.16	Common Driver	262
2.17	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	276
2.18	Lpc_freqme	297
2.19	LPCMP: Low Power Analog Comparator Driver	302
2.20	LPI2C: Low Power Inter-Integrated Circuit Driver	312
2.21	LPI2C Master Driver	313
2.22	LPI2C Master DMA Driver	328
2.23	LPI2C Slave Driver	330
2.24	LPSPi: Low Power Serial Peripheral Interface	341
2.25	LPSPi Peripheral driver	341
2.26	LPSPi eDMA Driver	362
2.27	LPTMR: Low-Power Timer	369
2.28	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	375

2.29	LPUART Driver	375
2.30	LPUART eDMA Driver	394
2.31	MCX_CMC: Core Mode Controller Driver	397
2.32	MCX_SPC: System Power Control driver	409
2.33	MCX_VBAT: Smart Power Switch	447
2.34	OPAMP: Operational Amplifier	467
2.35	PORT: Port Control and Interrupts	472
2.36	PWM: Pulse Width Modulator	479
2.37	Reset Driver	504
2.38	RTC: Real Time Clock	508
2.39	TRDC: Trusted Resource Domain Controller	515
2.40	Trdc_core	538
2.41	Trdc_soc	551
2.42	TRNG: True Random Number Generator	552
2.43	WUU: Wakeup Unit driver	557
2.44	WWDT: Windowed Watchdog Timer Driver	562
3	Middleware	567
3.1	Boot	567
3.1.1	MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource	567
3.1.2	MCUboot	568
3.2	File System	569
3.2.1	FatFs	569
3.3	Motor Control	571
3.3.1	FreeMASTER	571
4	RTOS	611
4.1	FreeRTOS	611
4.1.1	FreeRTOS kernel	611
4.1.2	FreeRTOS drivers	611
4.1.3	backoffalgorithm	611
4.1.4	corehttp	611
4.1.5	corejson	611
4.1.6	coremqtt	612
4.1.7	corepkcs11	612
4.1.8	freertos-plus-tcp	612

This documentation contains information specific to the frdm-mcxc162 board.

Chapter 1

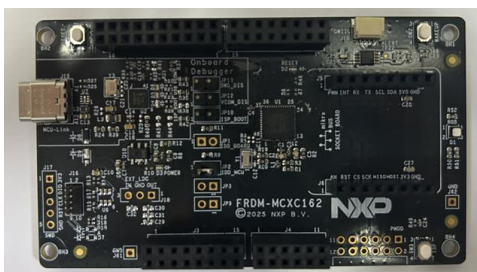
FRDM-MCXC162

1.1 Overview

FRDM-MCXC162 are compact and scalable development boards for rapid prototyping of MCX A57X MCUs. They offer industry standard headers for easy access to the MCUs I/Os, integrated open-standard serial interfaces and an on-board MCU-Link debugger. MCX A Series are high-performance, low-power microcontrollers with MAU, SmartDMA and performance efficiency.

The board is compatible with Arduino boards (Arduino UNO R3 and Arduino A4/A5), motor control boards (FRDM-MC-LVBLDC and FRDM-MC-LVPMSM), Mikroe click boards, and Pmod boards. It can be used with a wide range of development tools, including NXP MCUXpresso IDE, IAR Embedded Workbench, and Arm Keil MDK. The board is lead-free and RoHS-compliant.

For debugging the MCXC162 MCU, the FRDM-MCXC162 board uses an onboard (OB) debug probe, MCU-Link lite OB, which is based on another NXP MCU: LPC55S16



MCU device and part on board is shown below:

- Device: MCXC162
- PartNumber: MCXC162VFT

1.2 Getting Started with MCUXpresso SDK Package

1.2.1 Getting Started with MCUXpresso SDK Package

Starting with version 25.09.00, MCUXpresso SDK introduced two package versions for offline development:

- **Classic SDK Package:** Traditional board-specific packages with pre-configured IDE projects for MCUXpresso IDE, IAR, Keil, and other toolchains.
- **Repository-Layout SDK Package:** Board-specific packages that maintain the same structure and build system as the GitHub Repository SDK, providing offline access to

the repository SDK development experience. Available when selecting the ARMGCC toolchain.

From version 25.12.00 onward:

- When you select ARMGCC, the SDK download will use the Repository-Layout version.
- For all other toolchains, the SDK download will remain in the Classic version.

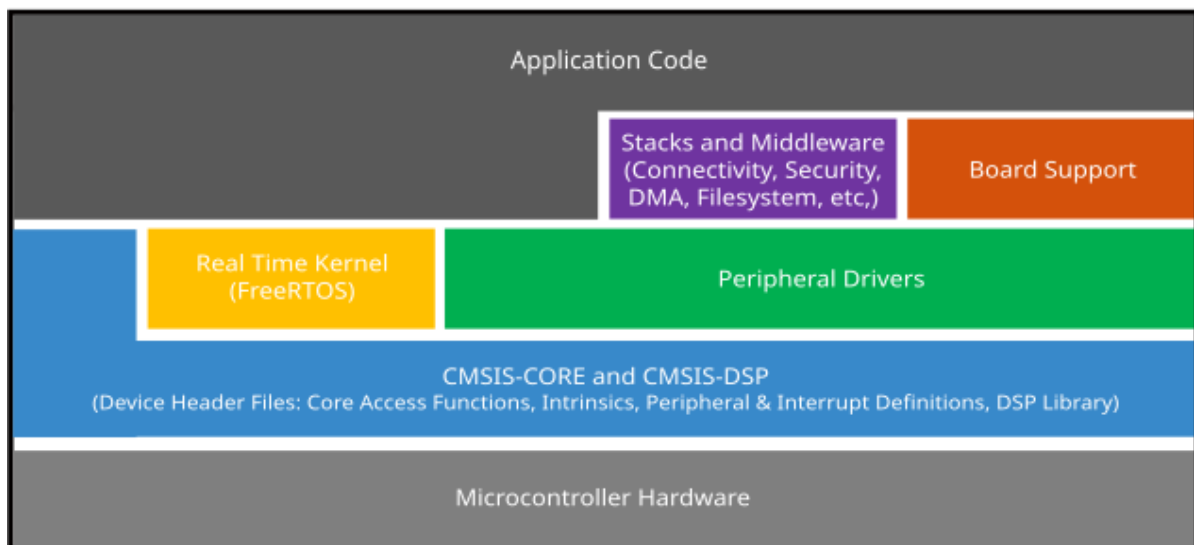
Note: The Repository-Layout SDK package was first introduced in version 25.09.00, but initially only for MCXW23x platforms.

Classic 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):

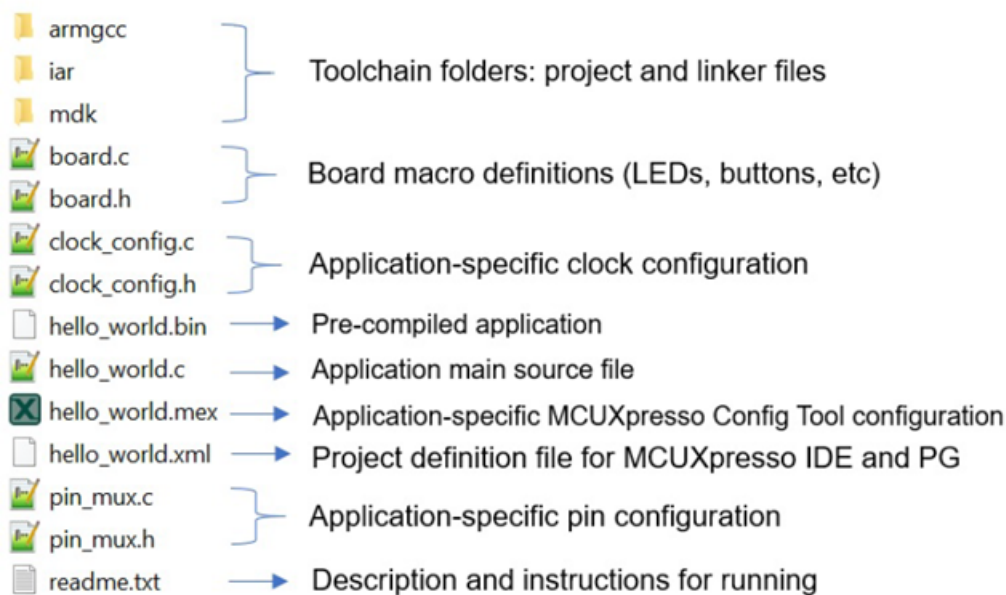
- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.

- **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.

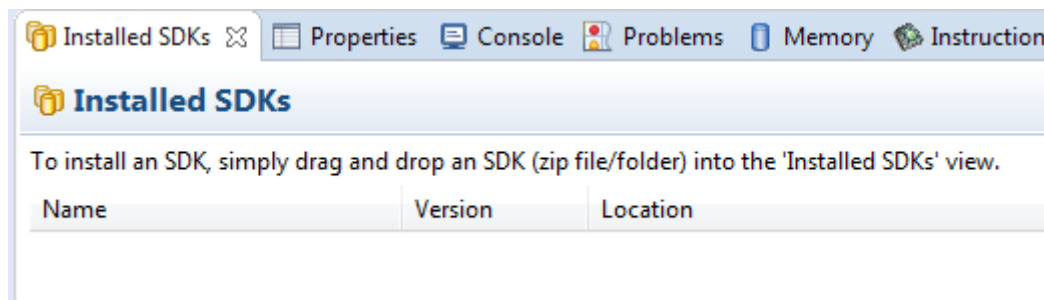
Run a demo using MCUXpresso IDE **Note:** Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

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.

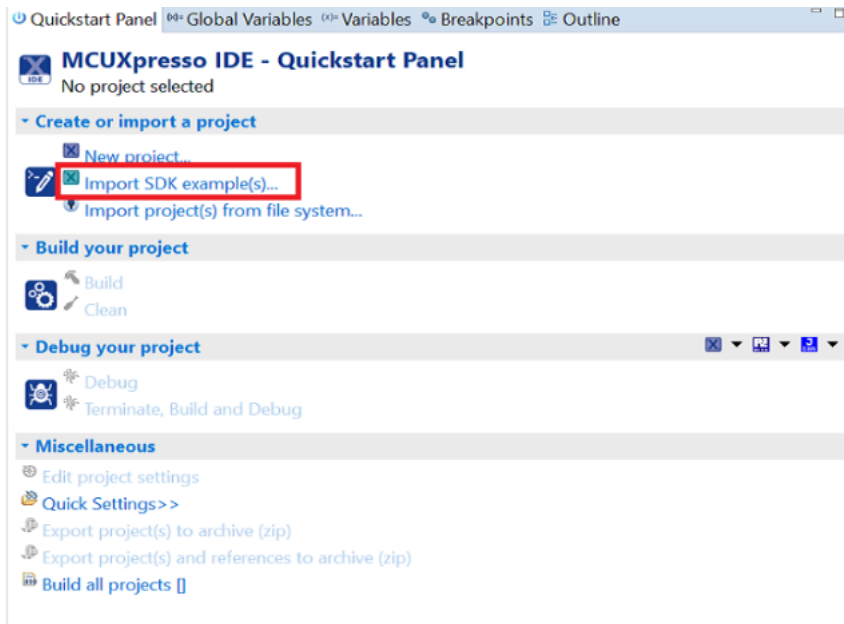
Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. 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.

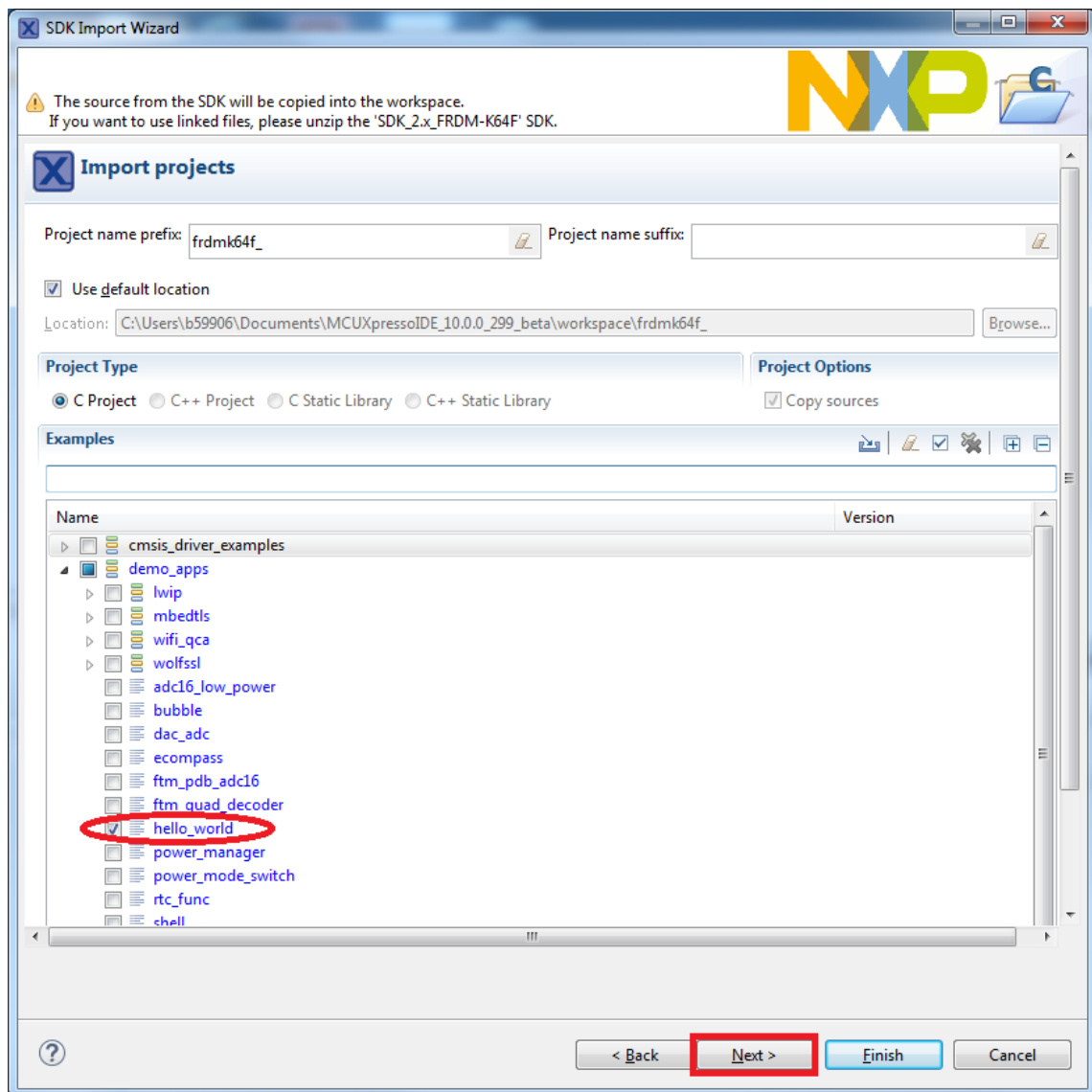
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



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



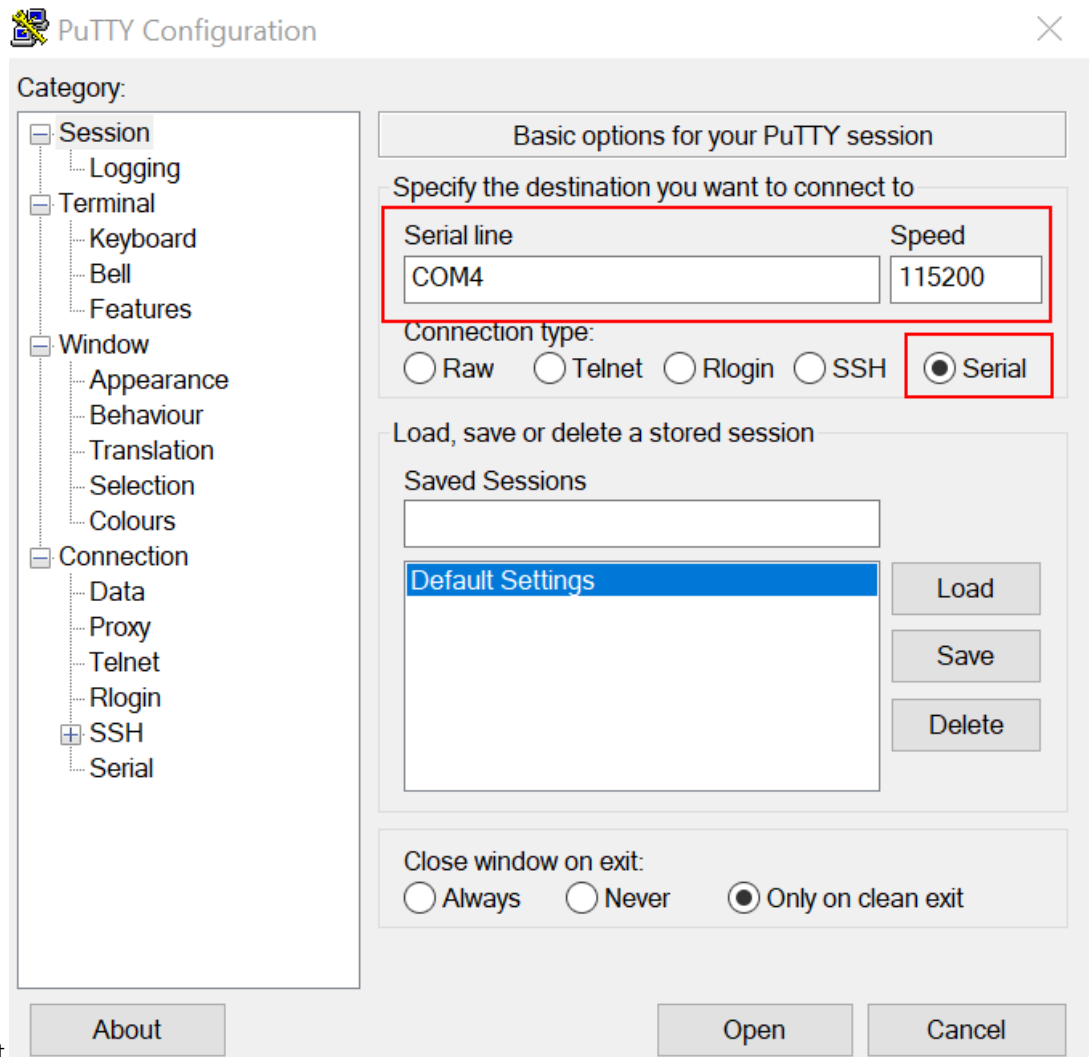
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

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:

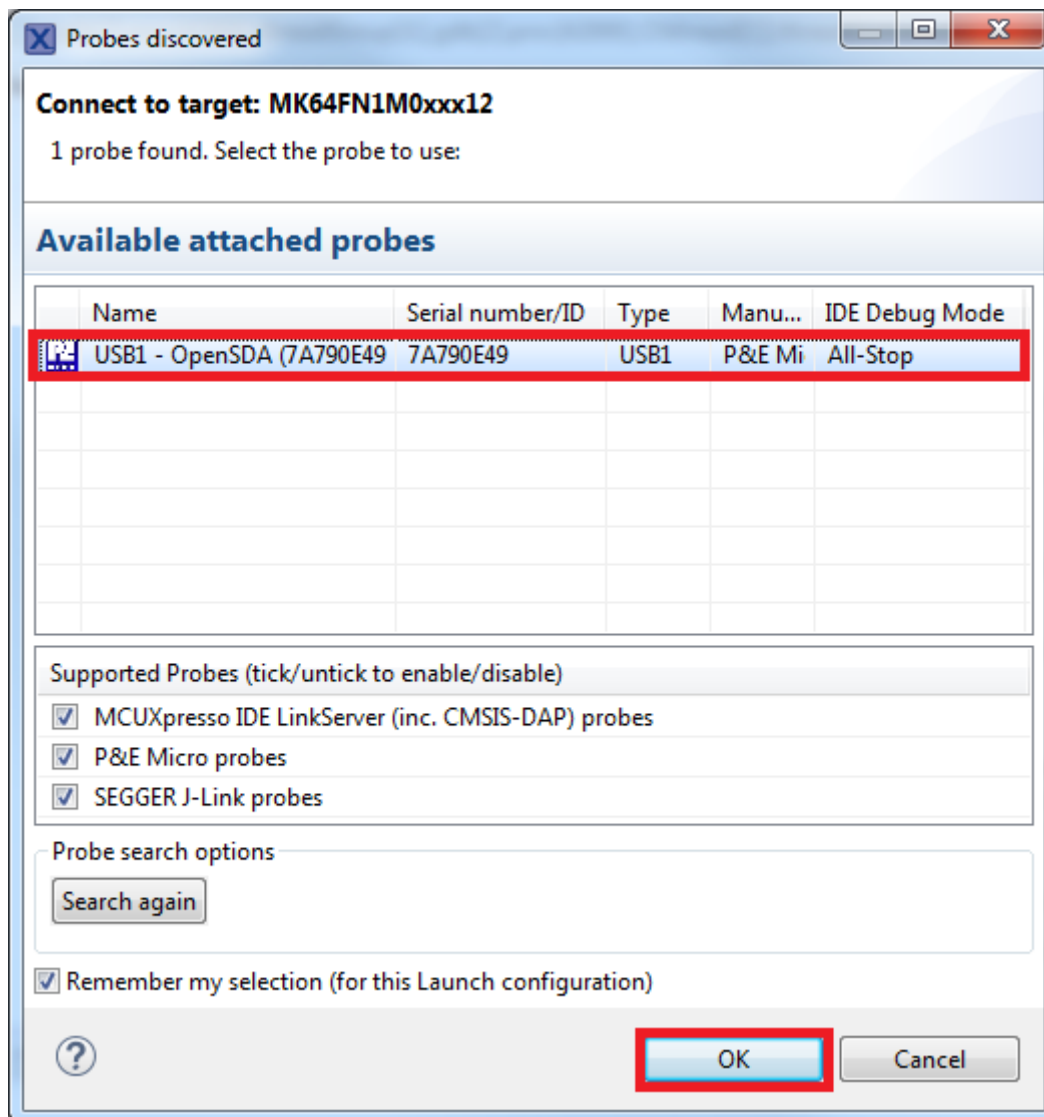
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM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

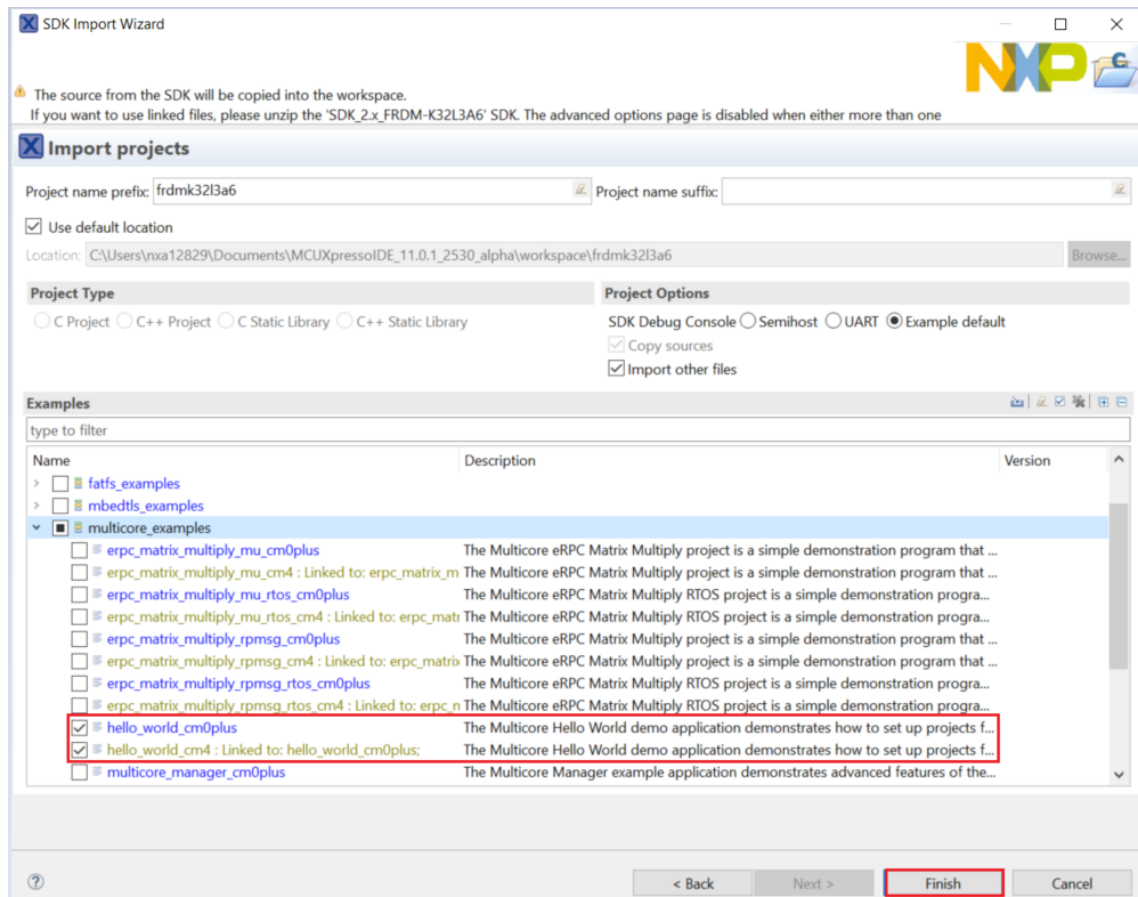


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

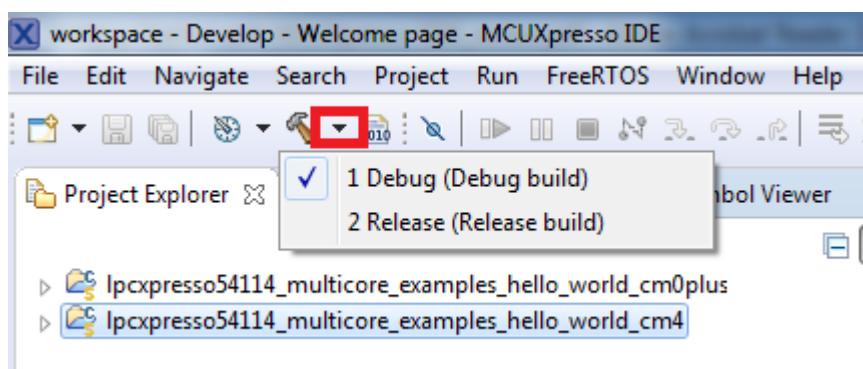


Build a multicore example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world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

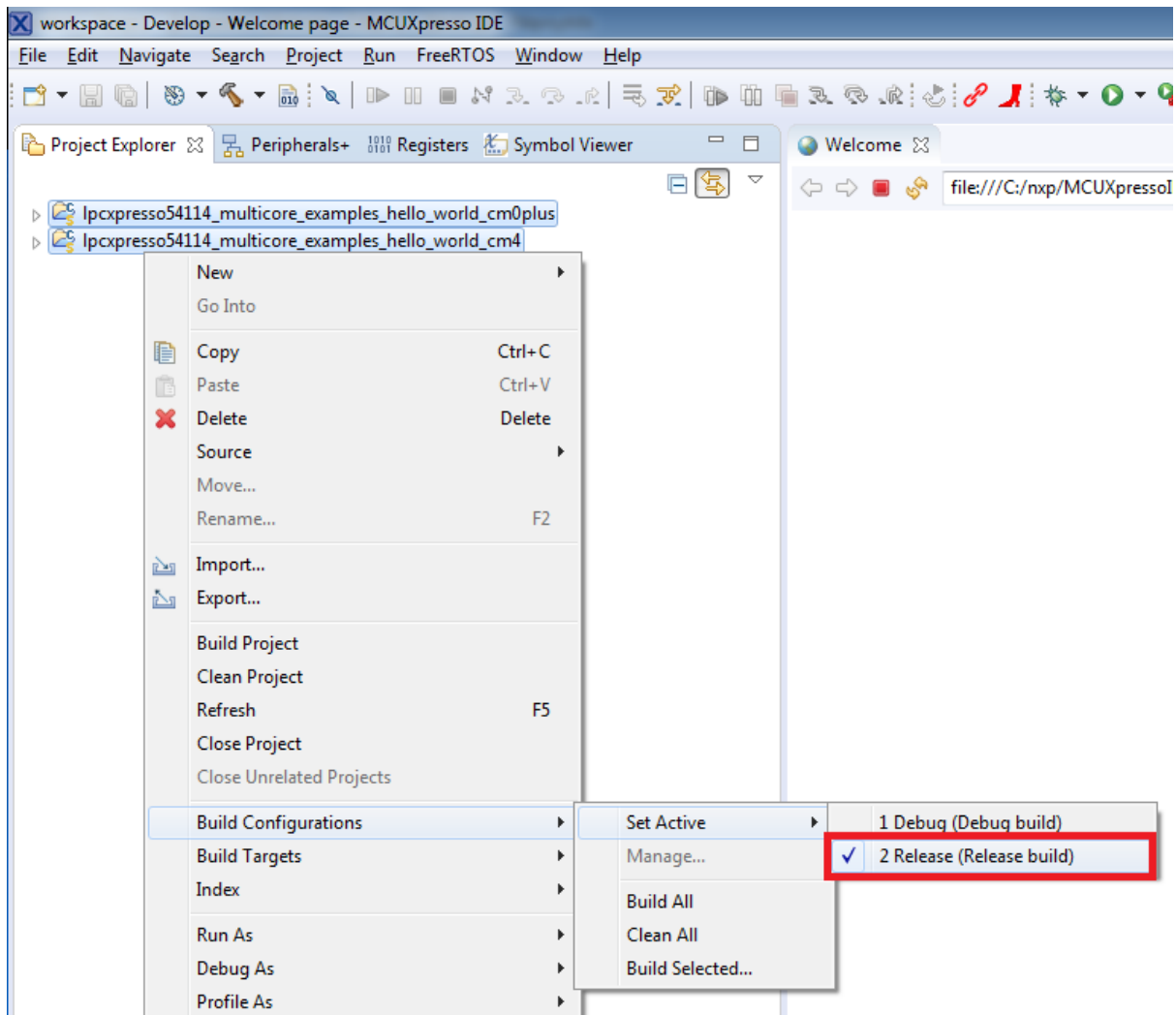


3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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

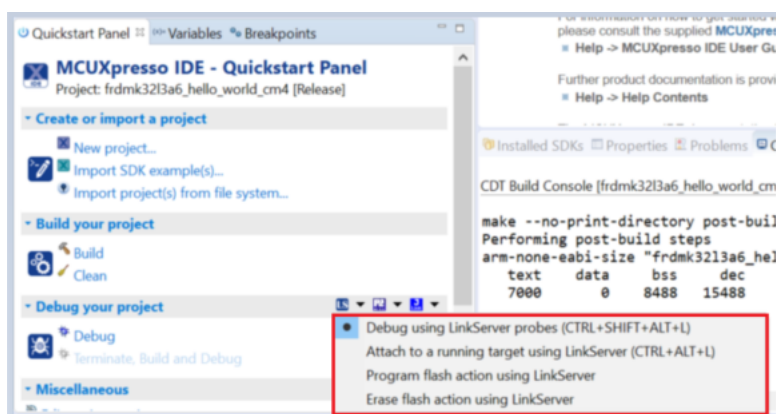


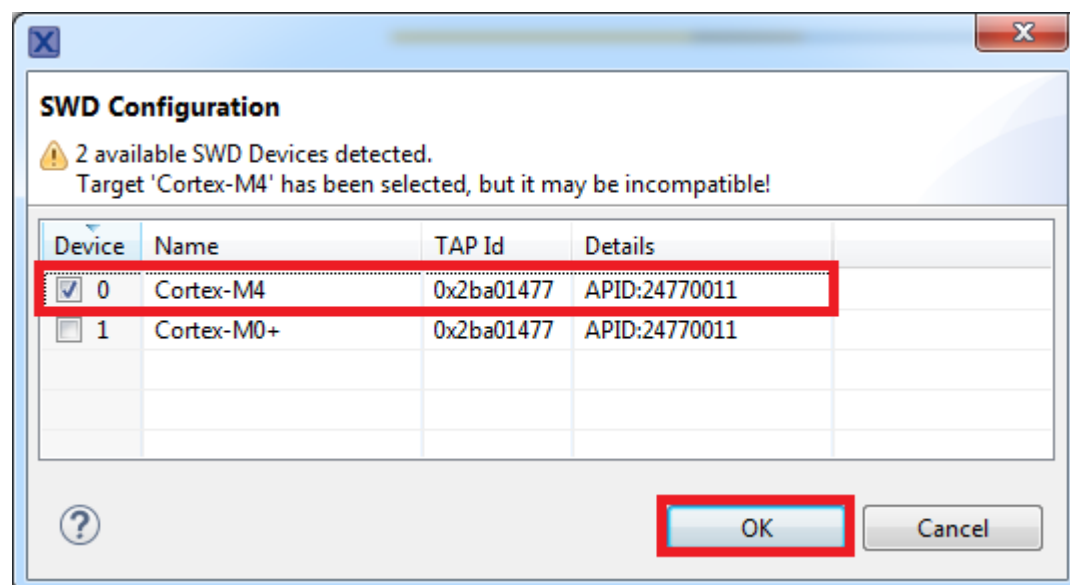
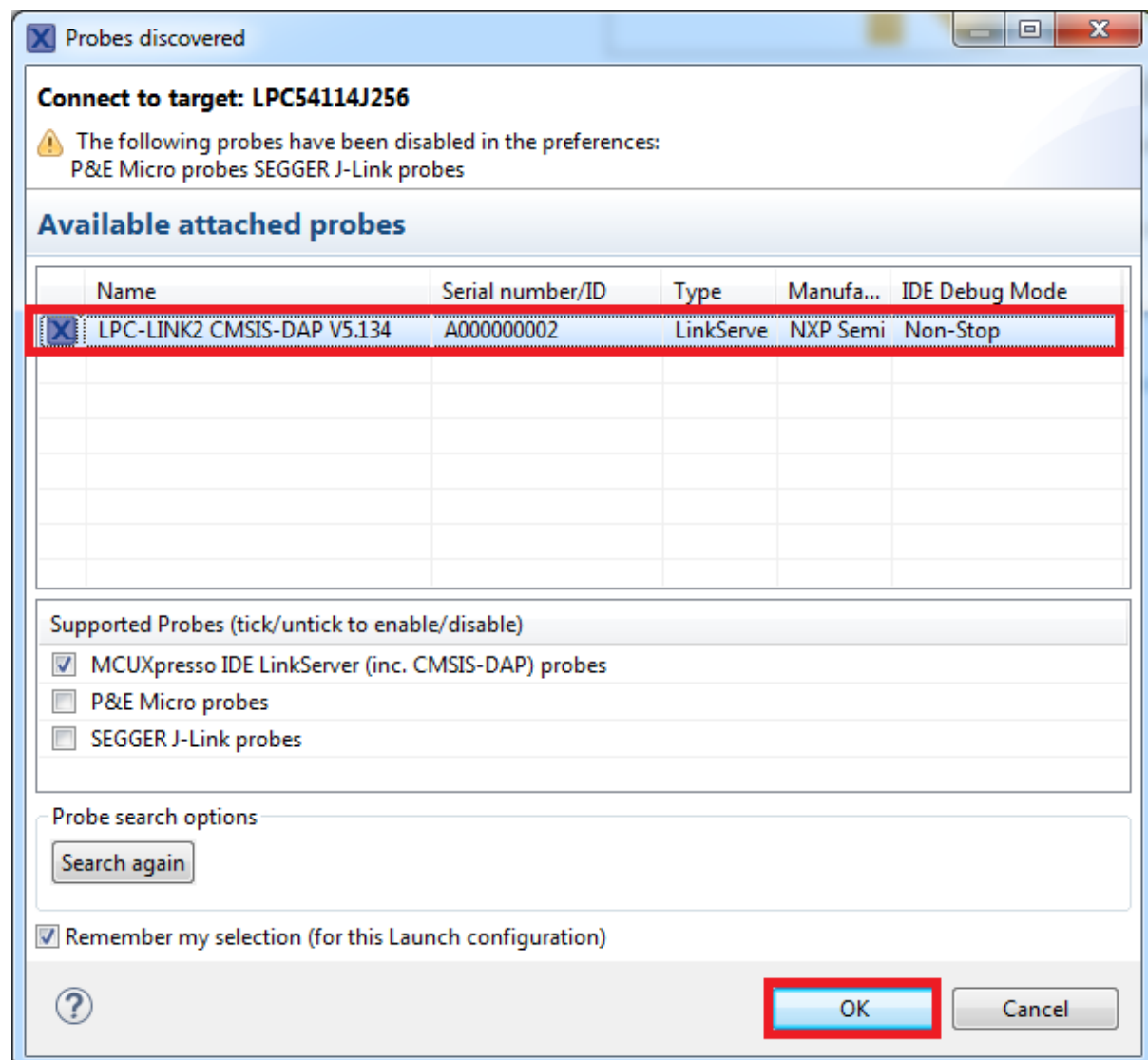
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

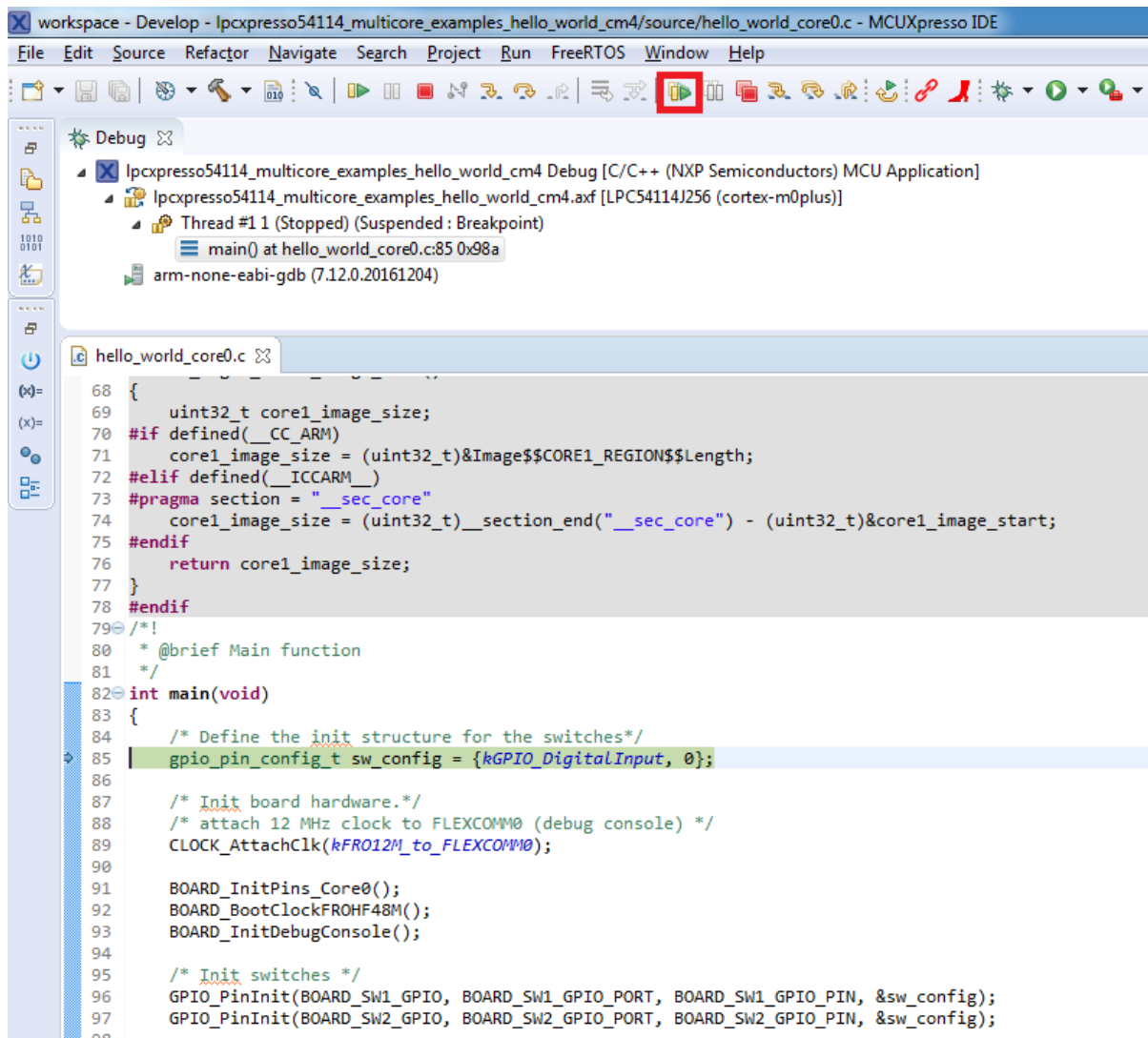
Note: When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build 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.



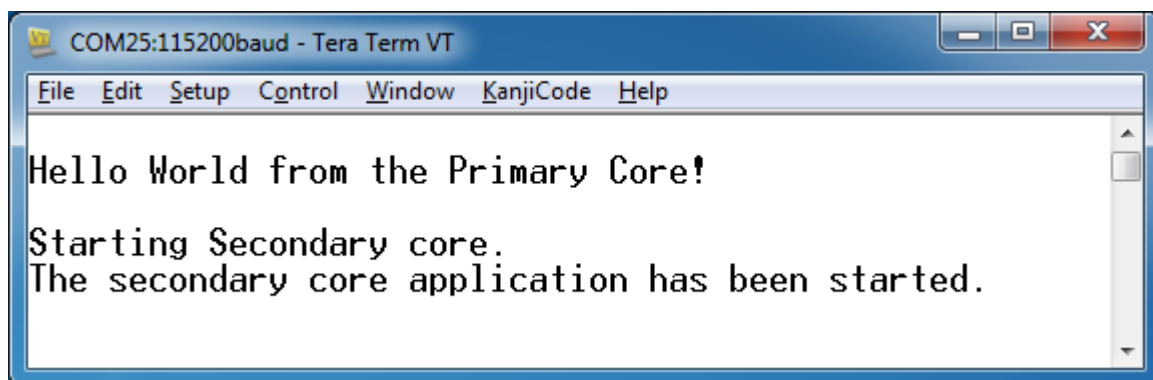
Run a multicore example application The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run an 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.





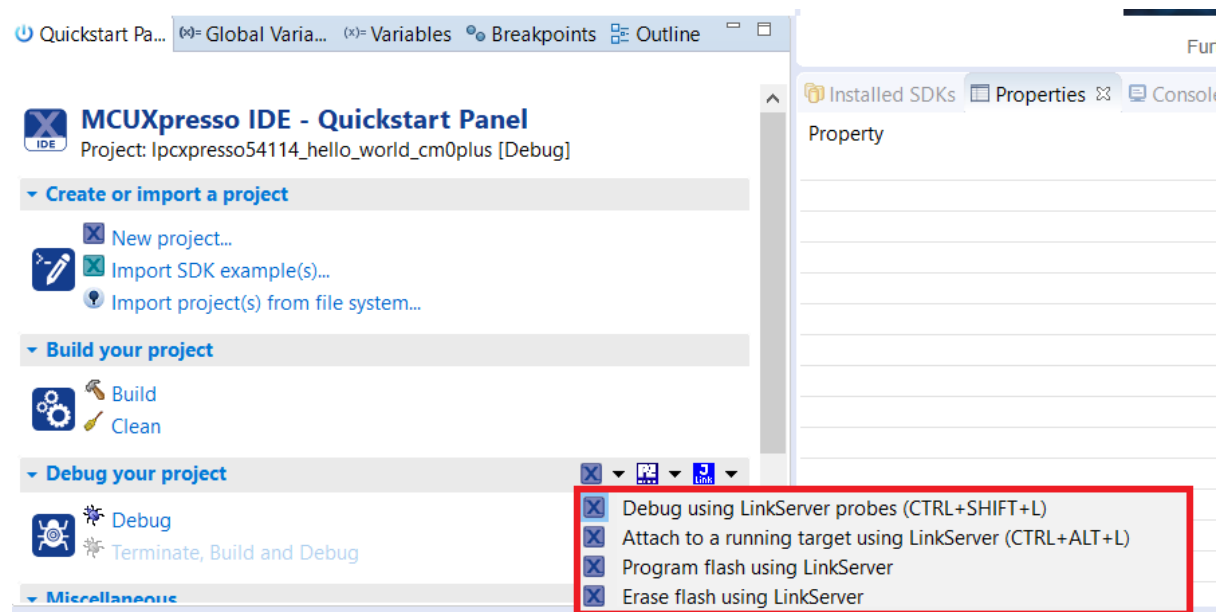


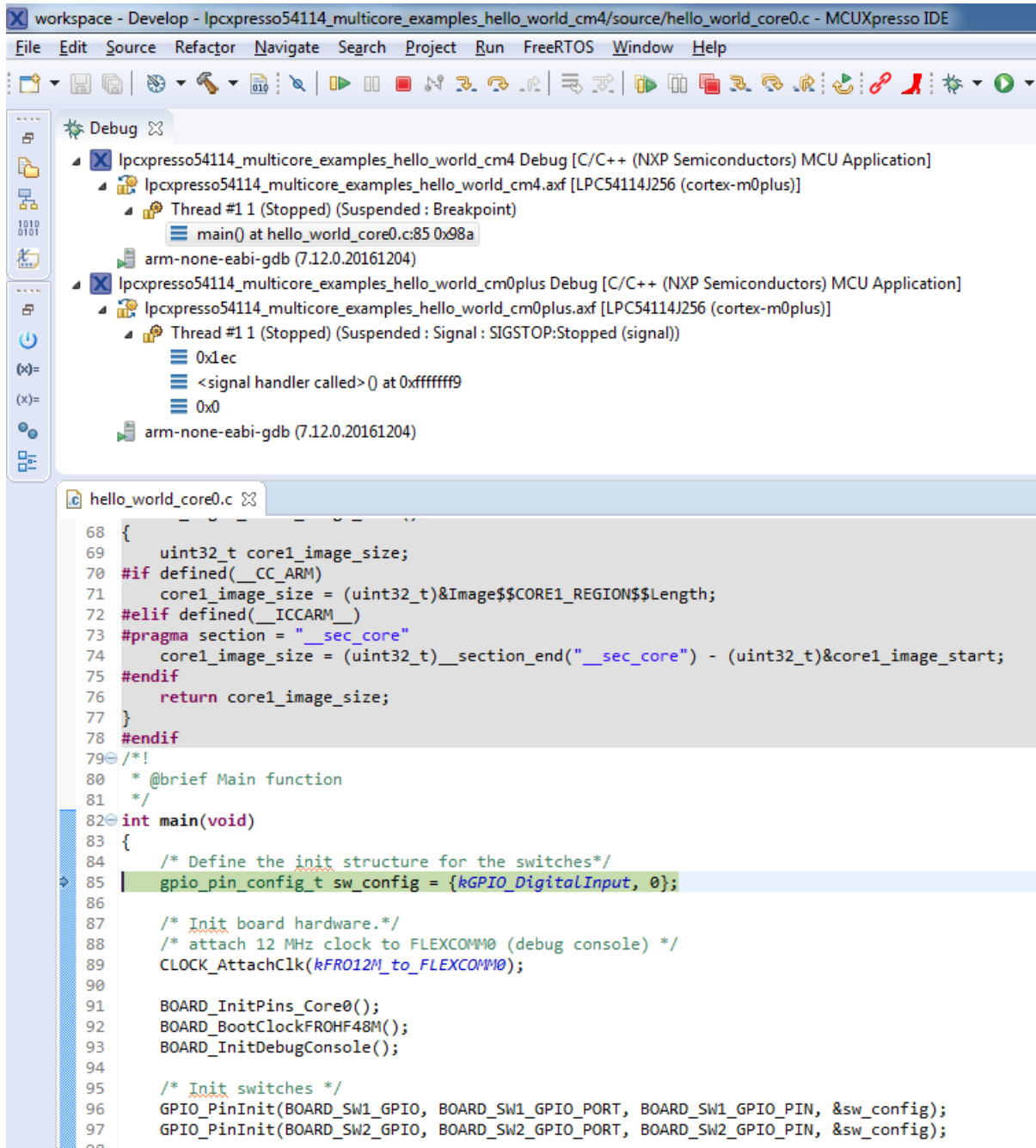
After clicking the “Resume All Debug sessions” button, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



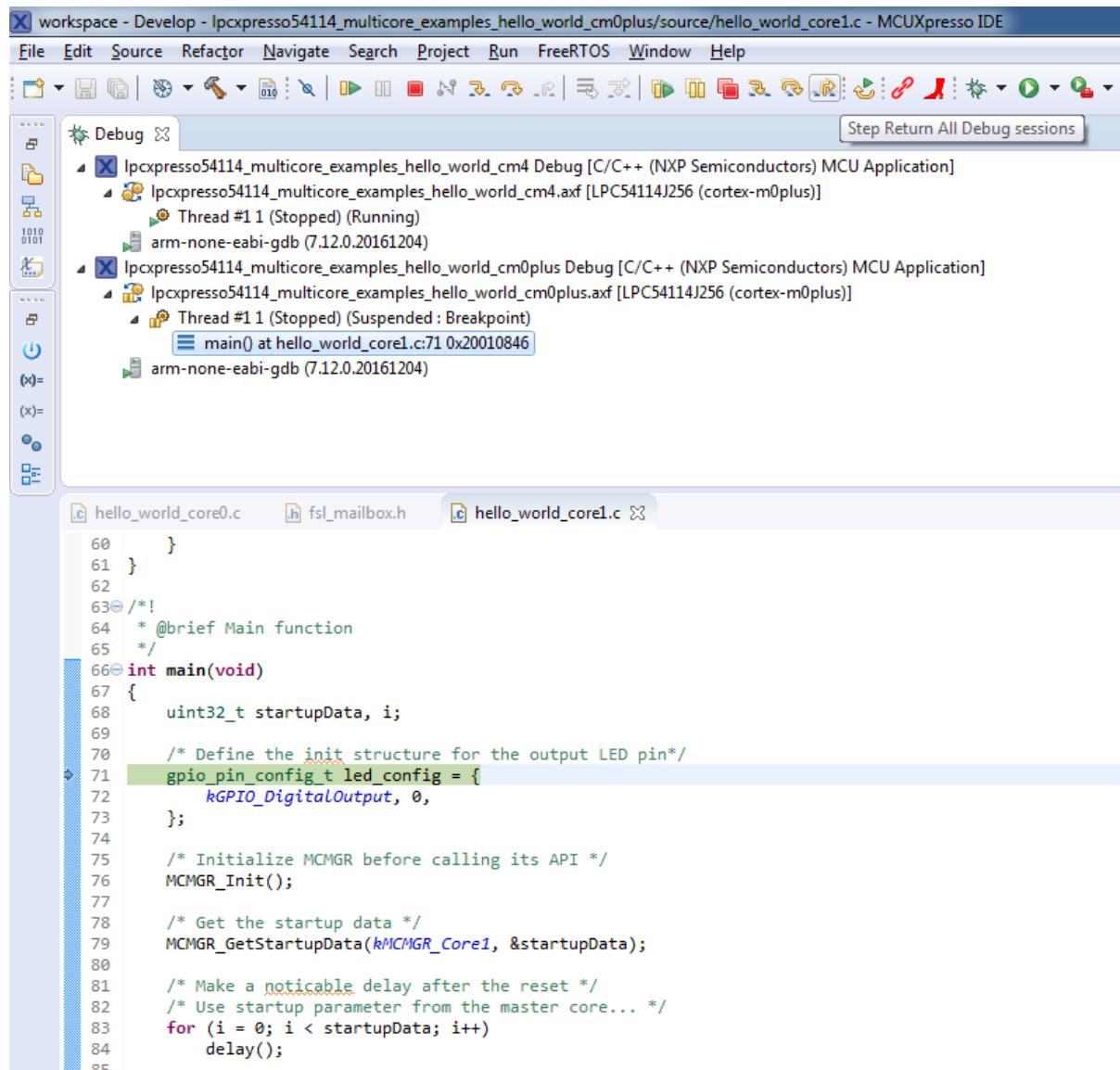
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps 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.

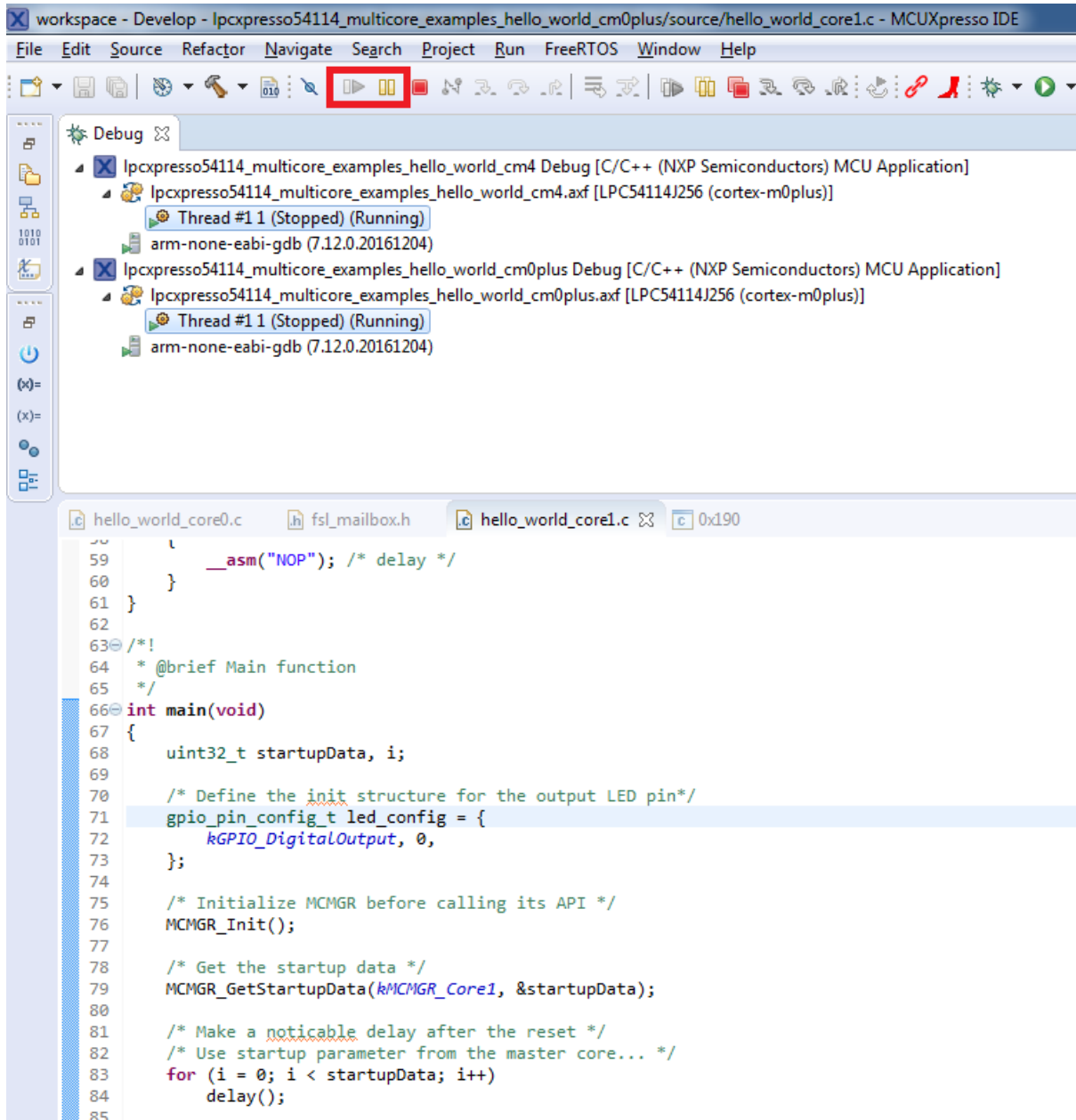


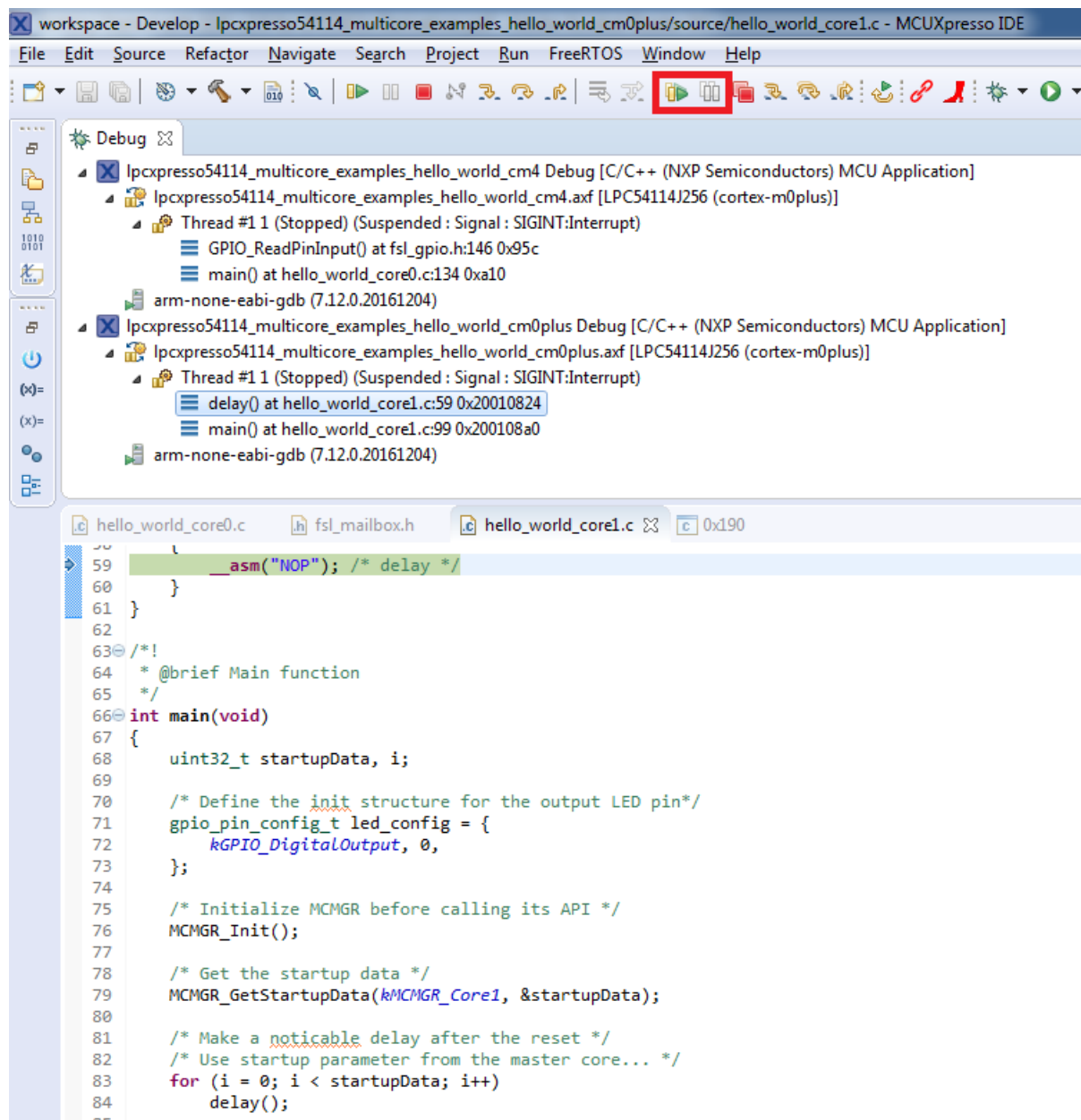


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello_world multicore application then starts running. The primary core 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.



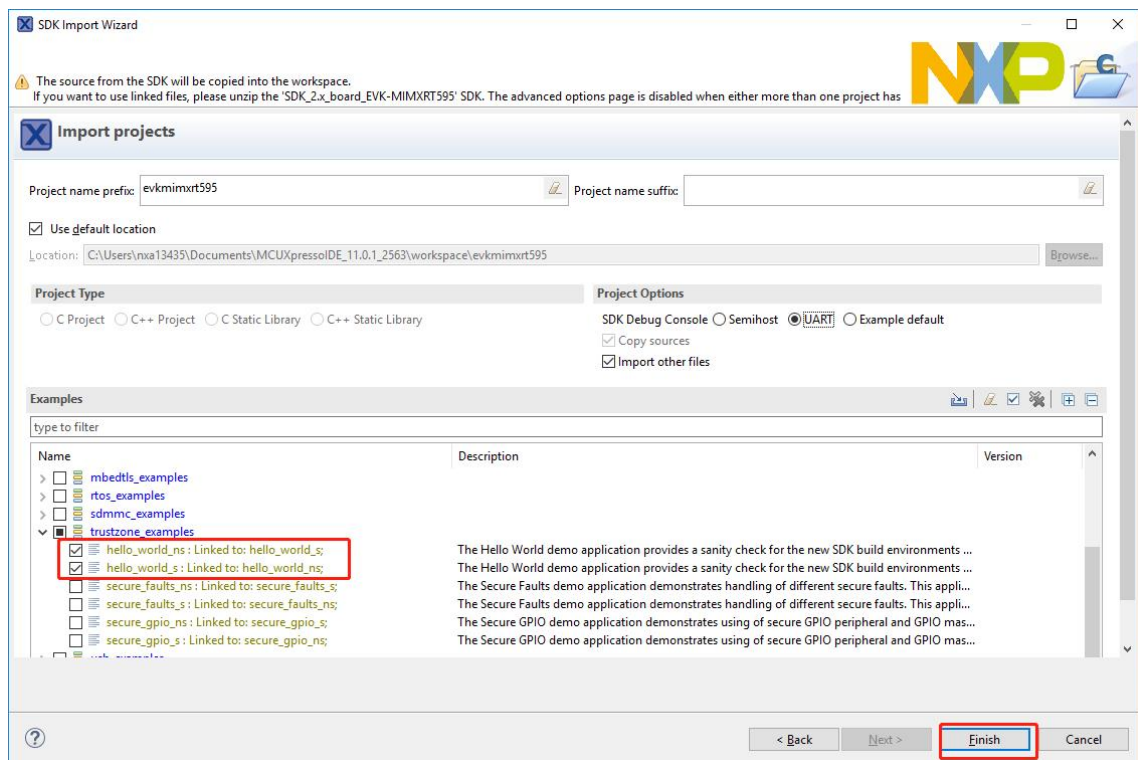
At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both the cores. This is done either by selecting both opened debug sessions (multiple selections) and clicking the “Suspend” / “Resume” control button,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



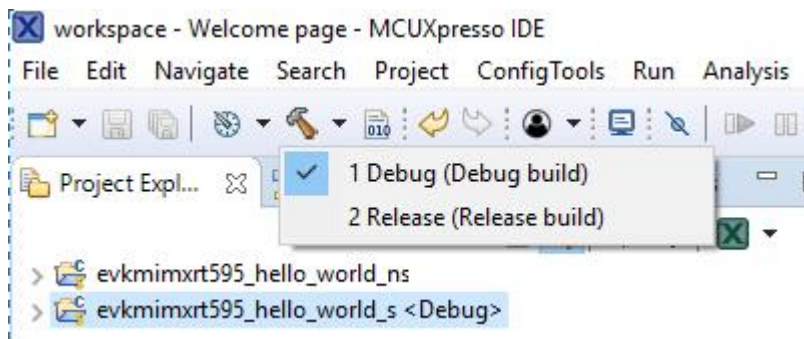


Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the hello_world example application targeted for the MIMXRT595-EVK hardware platform is used as an example, though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

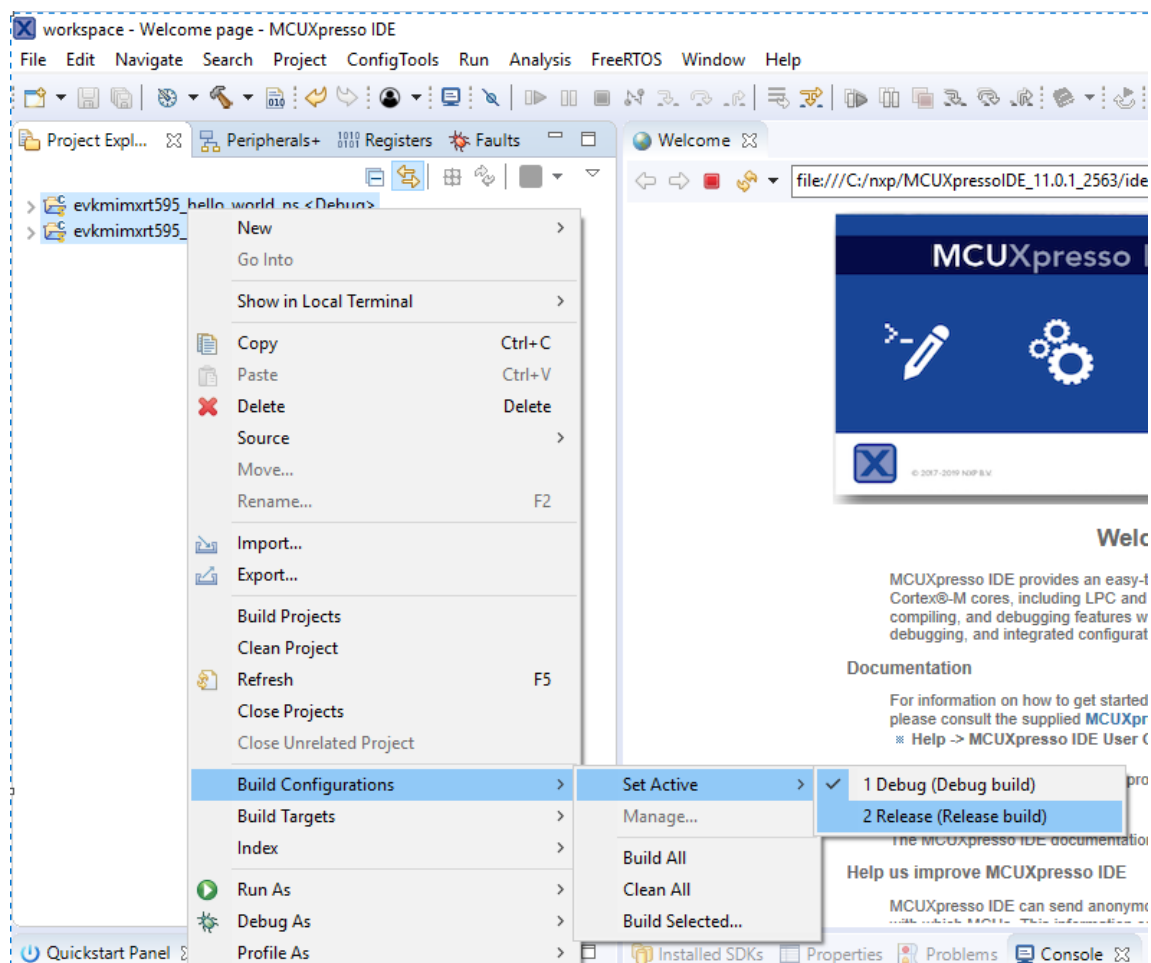


3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward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



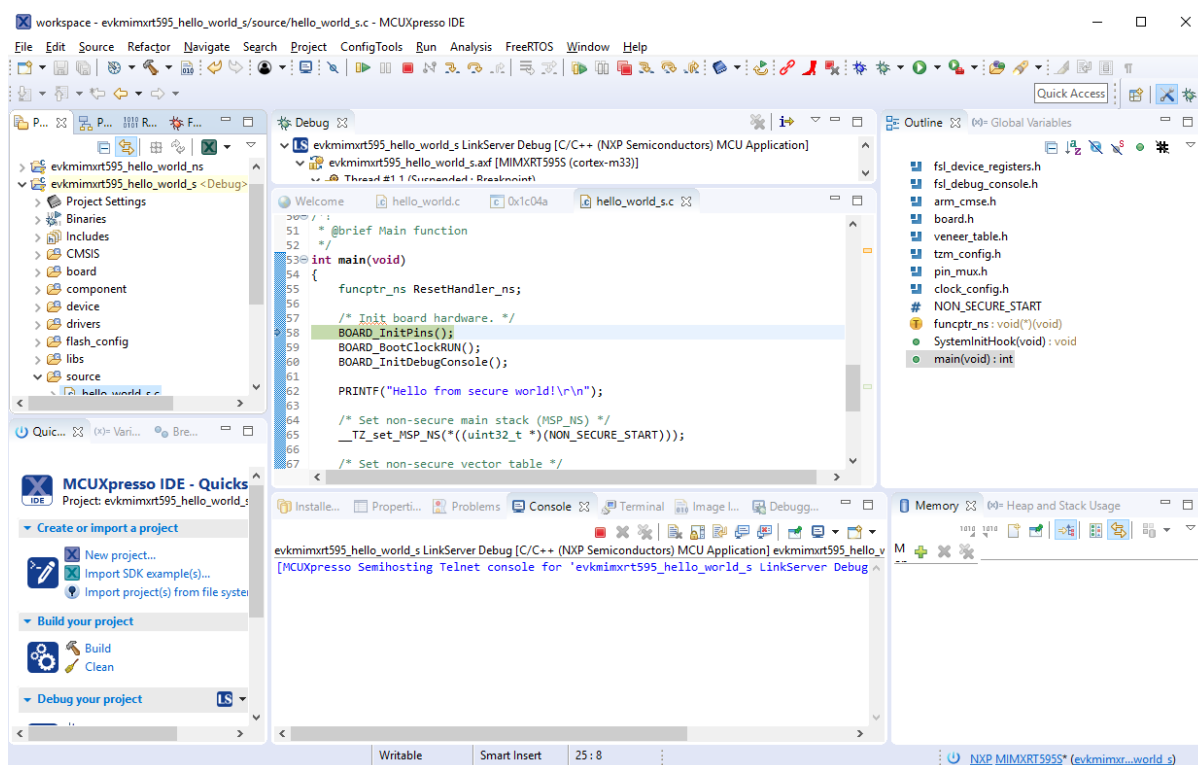
The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when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

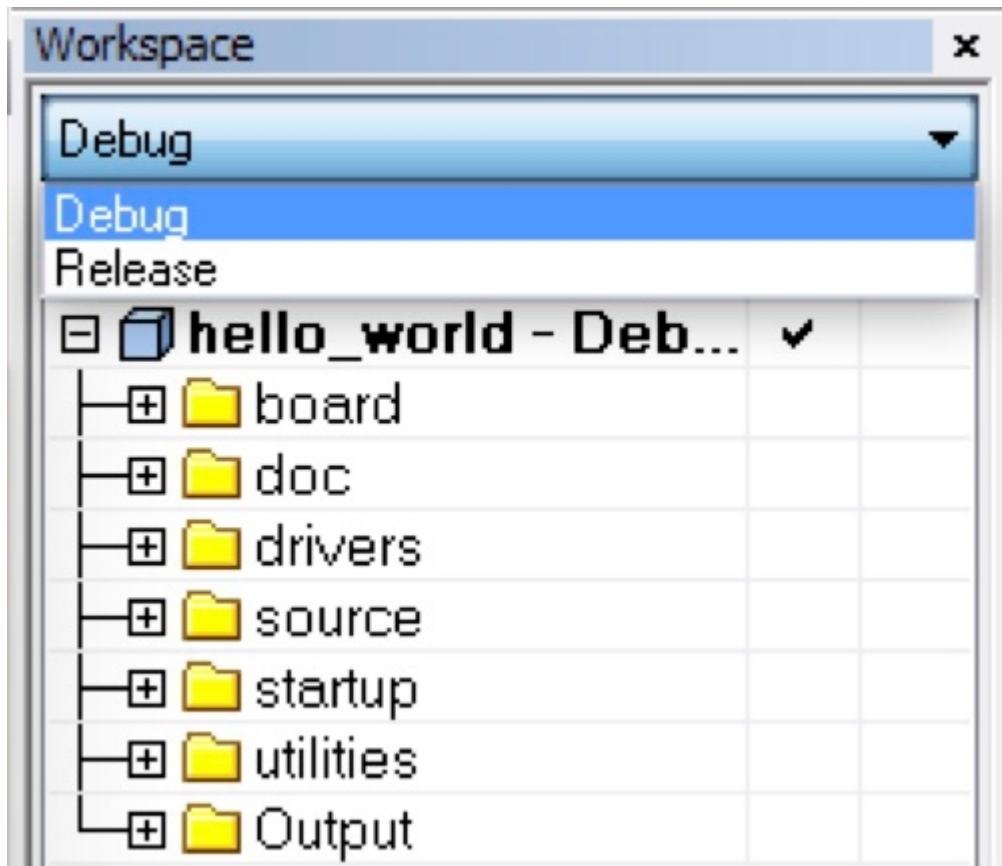
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

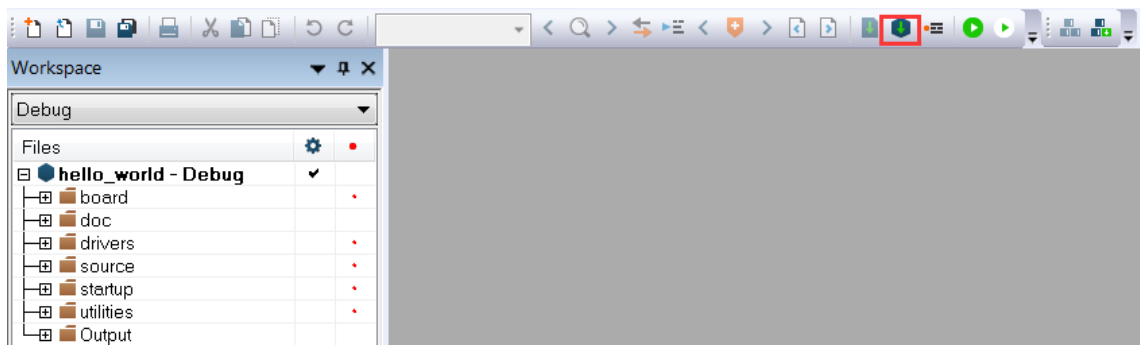
Other example applications may have additional folders in their path.

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

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



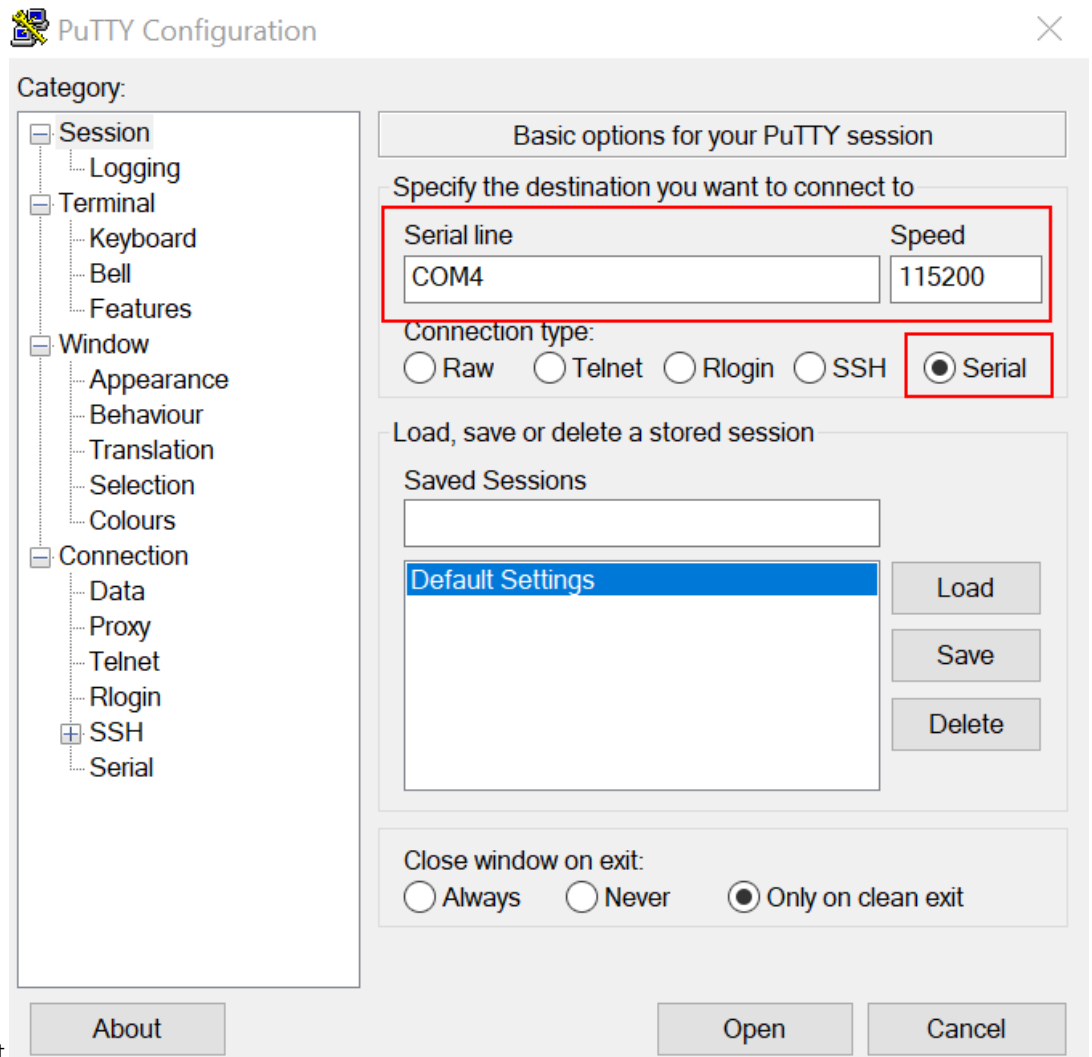
3. To build the demo application, click **Make**, highlighted in red in following figure.



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

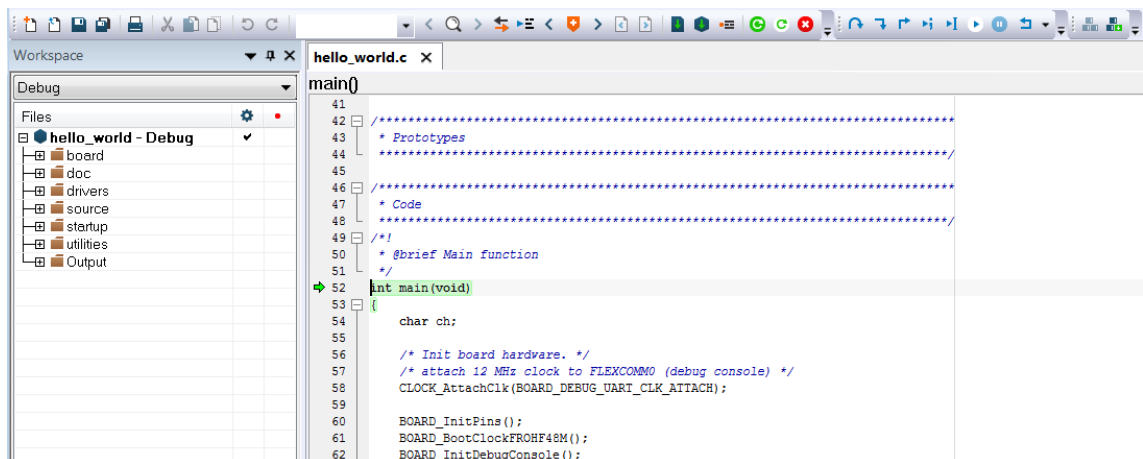


4. 1 stop bit

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
```

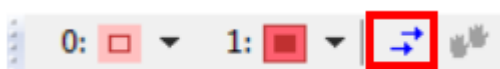
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

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.

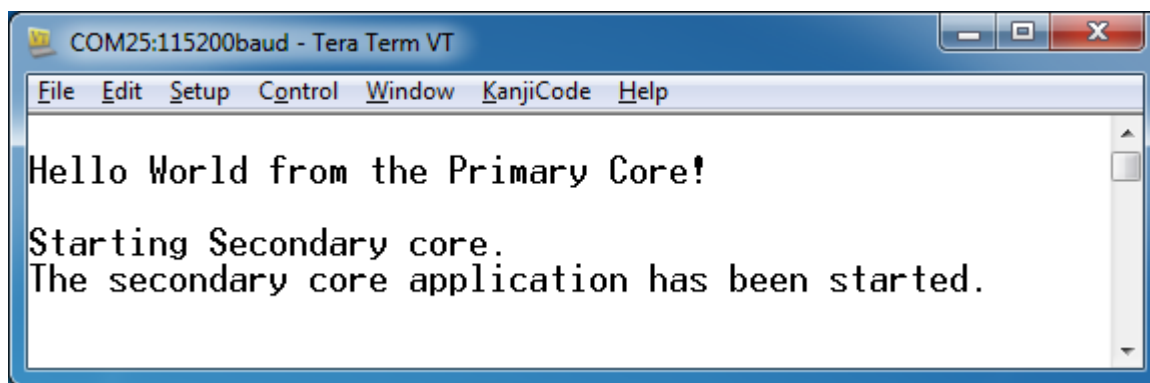
Run a multicore example application The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an example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

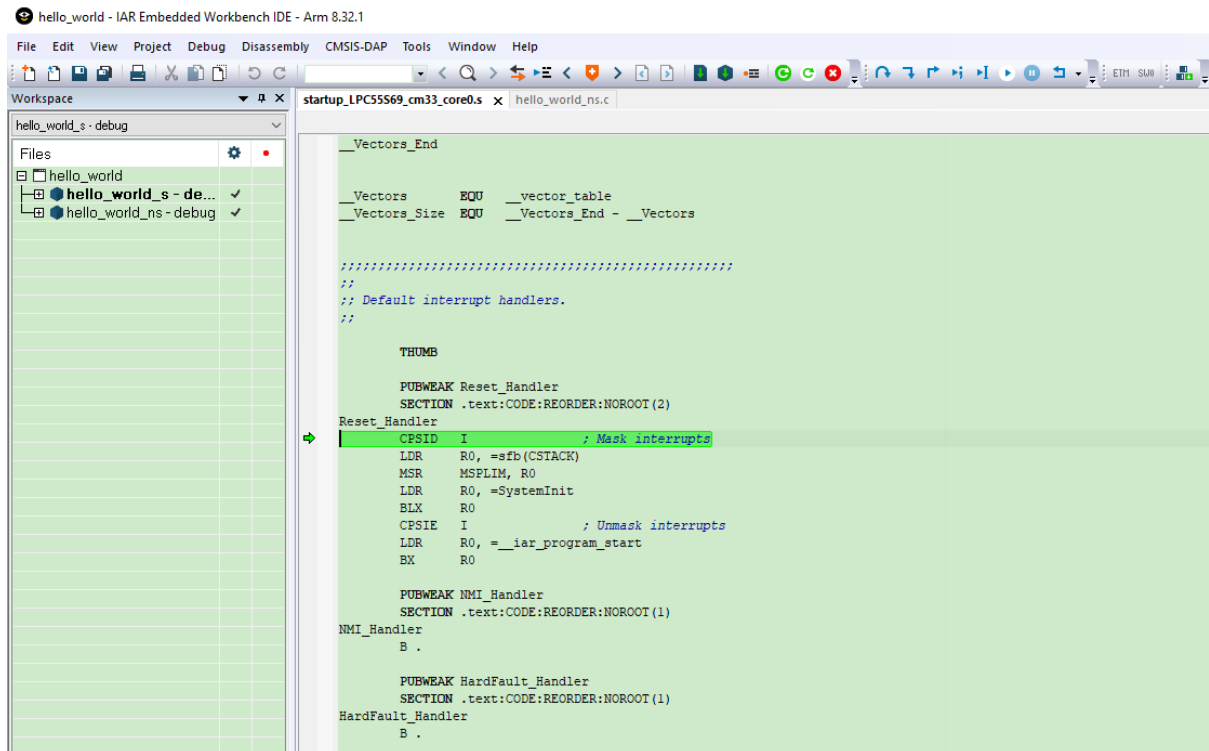
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_  
↪ns.eww
```

```
<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
```

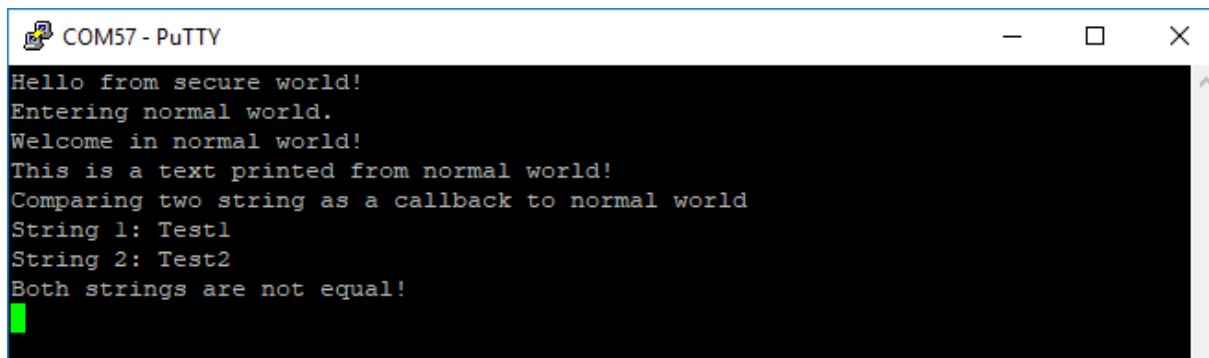
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because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

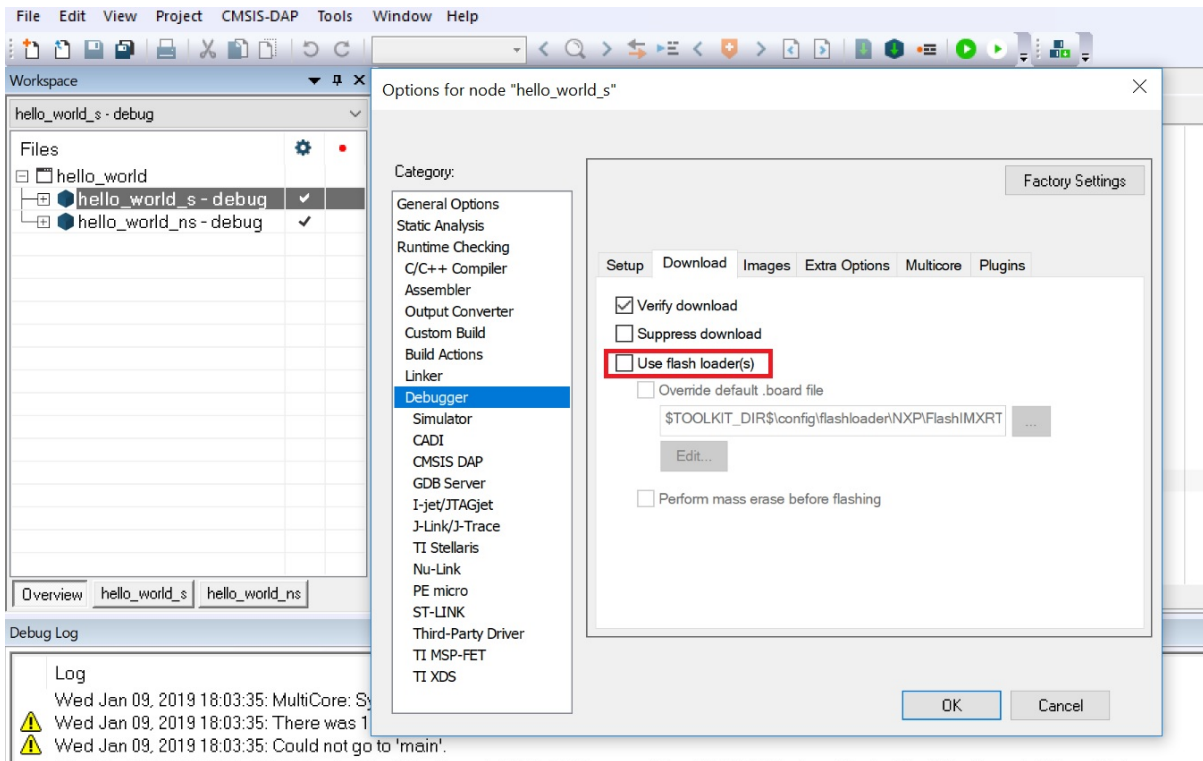


Run the code by clicking **Go** to start the application.

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.



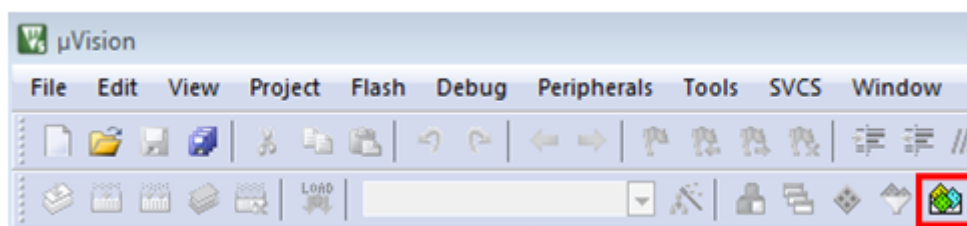
Note: If the application is running in RAM (debug/release build target), in **Options**>**Debugger > Download** tab, disable **Use flash loader(s)**. This can avoid the __ns download issue on i.MXRT500.



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:

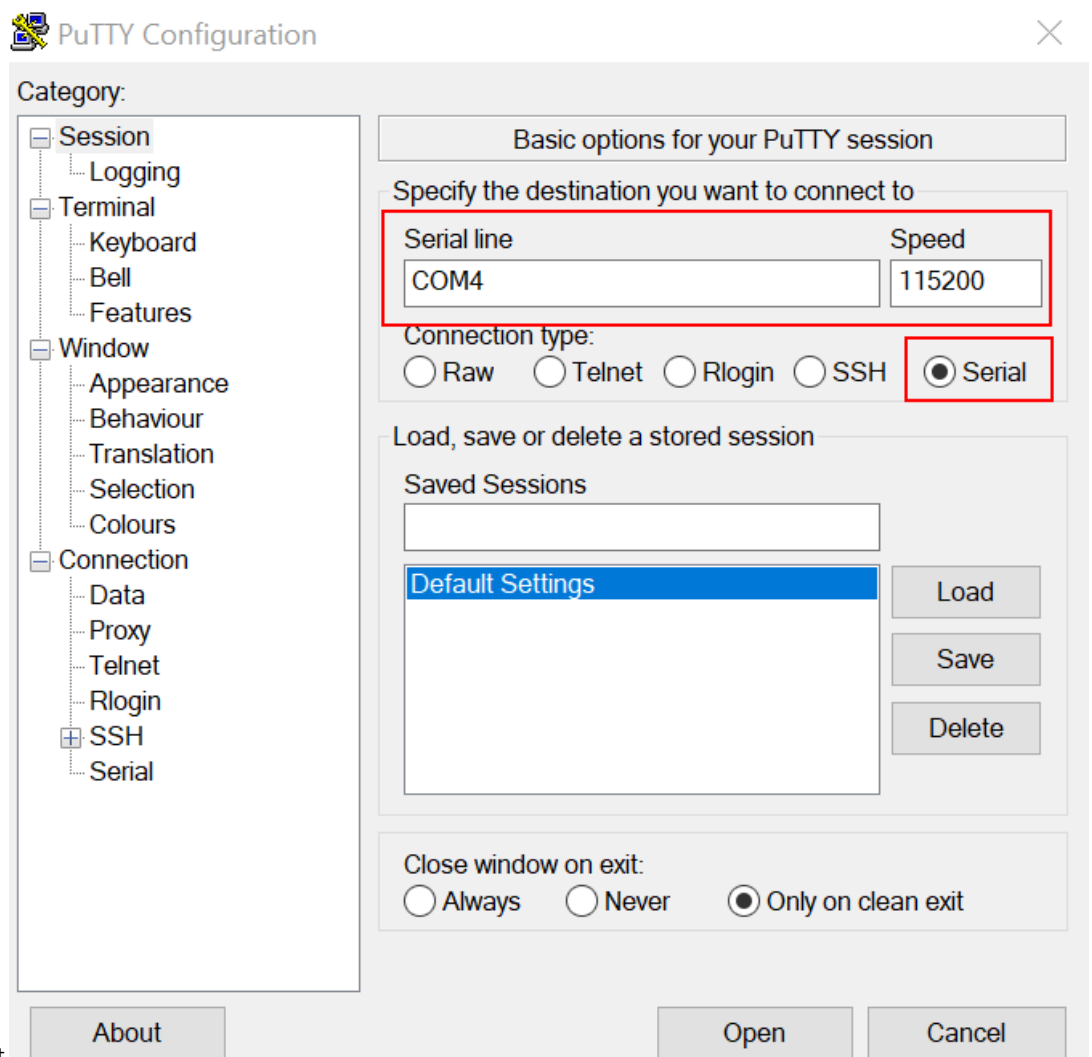
2. To build the demo project, select **Rebuild**, highlighted in red.



3. The build completes without errors.

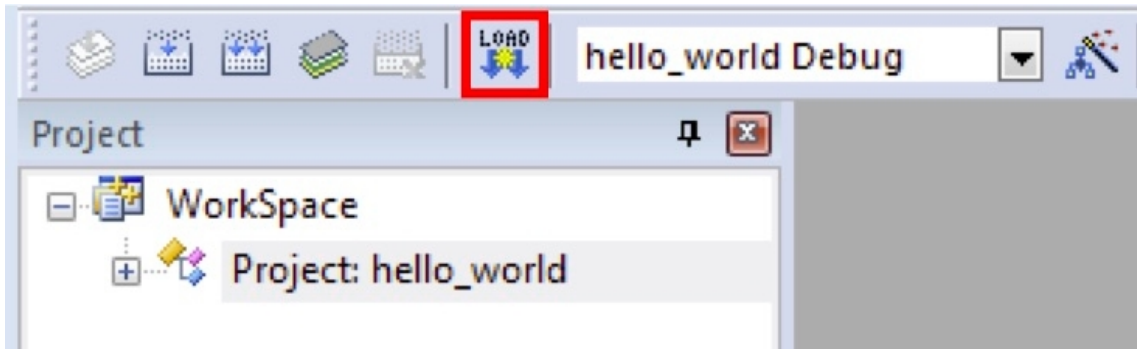
Run an example application To download and run the application, perform these steps:

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 using USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine the COM port number, see [How to determine COM port](#)). Configure the terminal with these settings:
 1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
 2. No parity
 3. 8 data bits

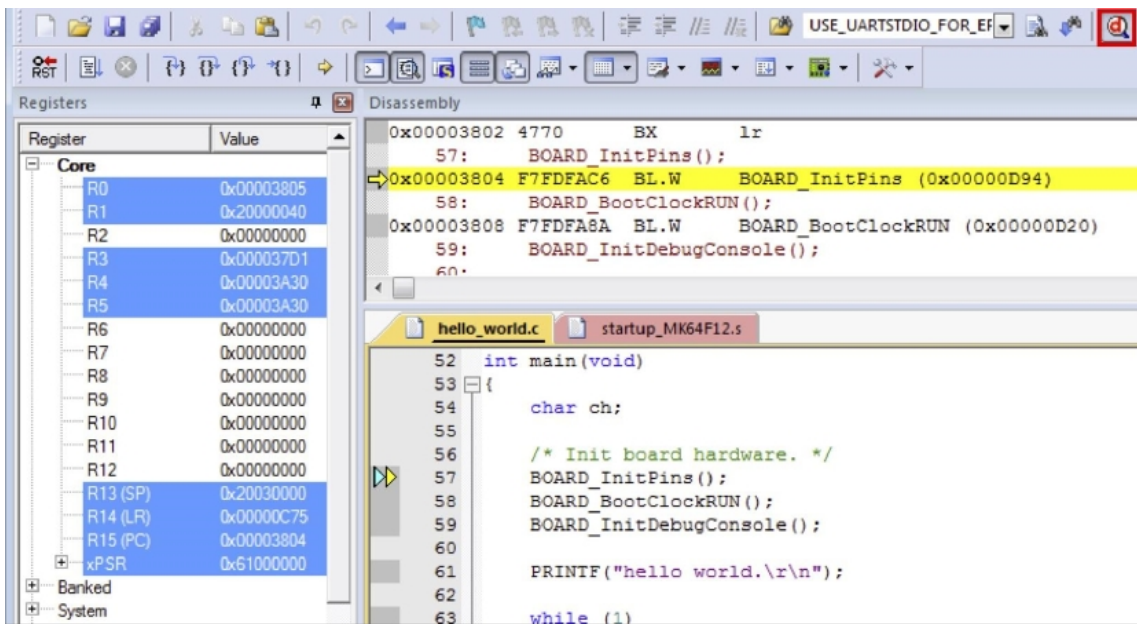


4. 1 stop bit

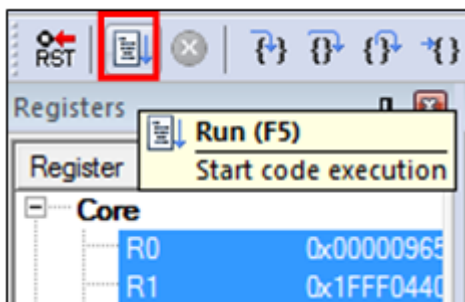
4. In μ Vision, after the application is built, click the **Download** button to download the application to the target.



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/ μ Vision workspaces are located in this folder:

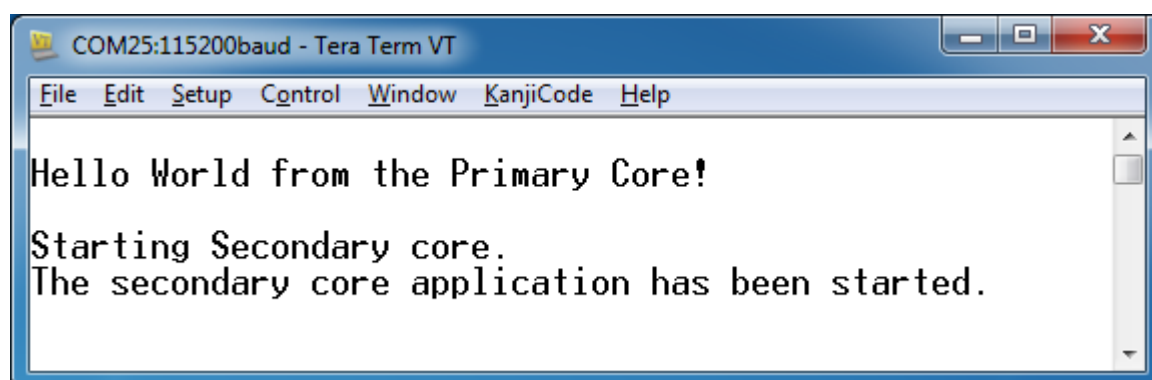
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
→cm0plus.uvmpw
```

```
<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.

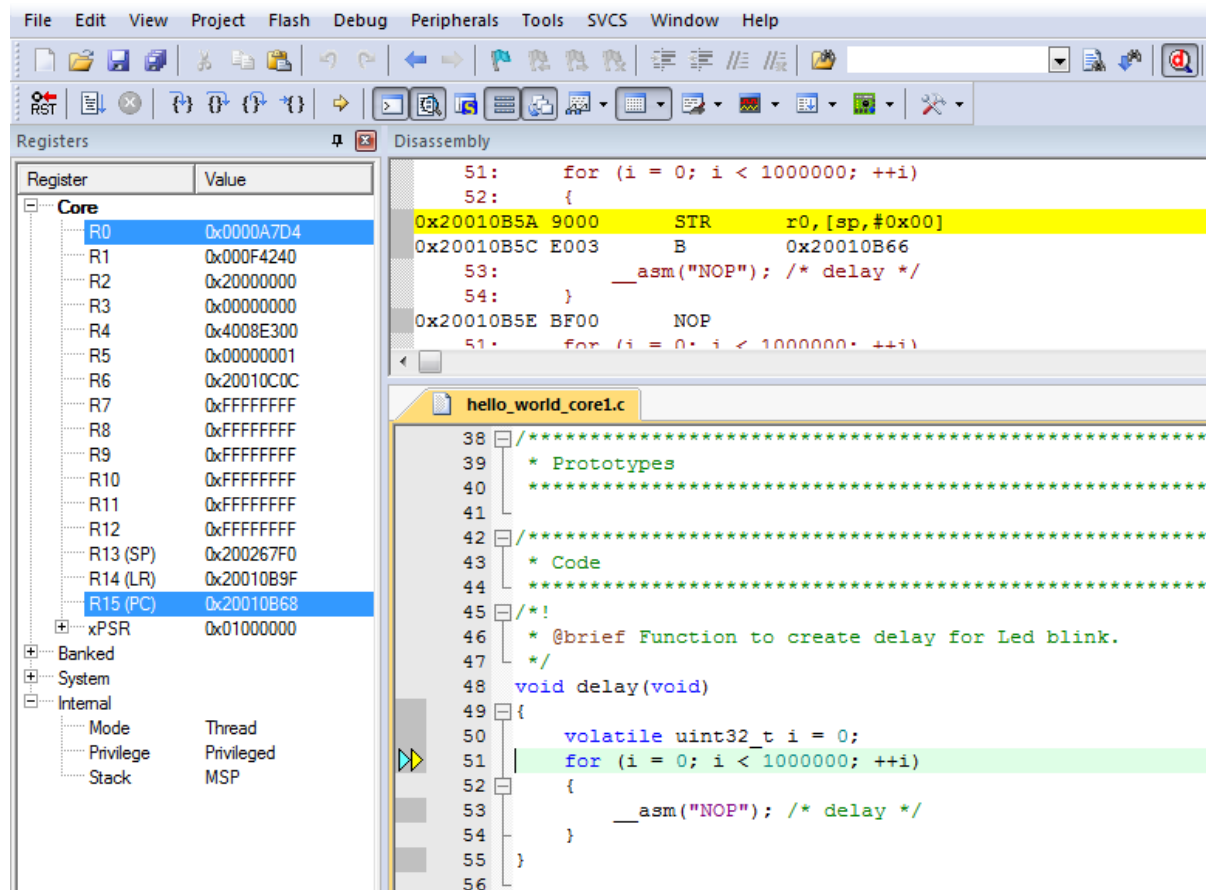
Run a multicore example application The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

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
```

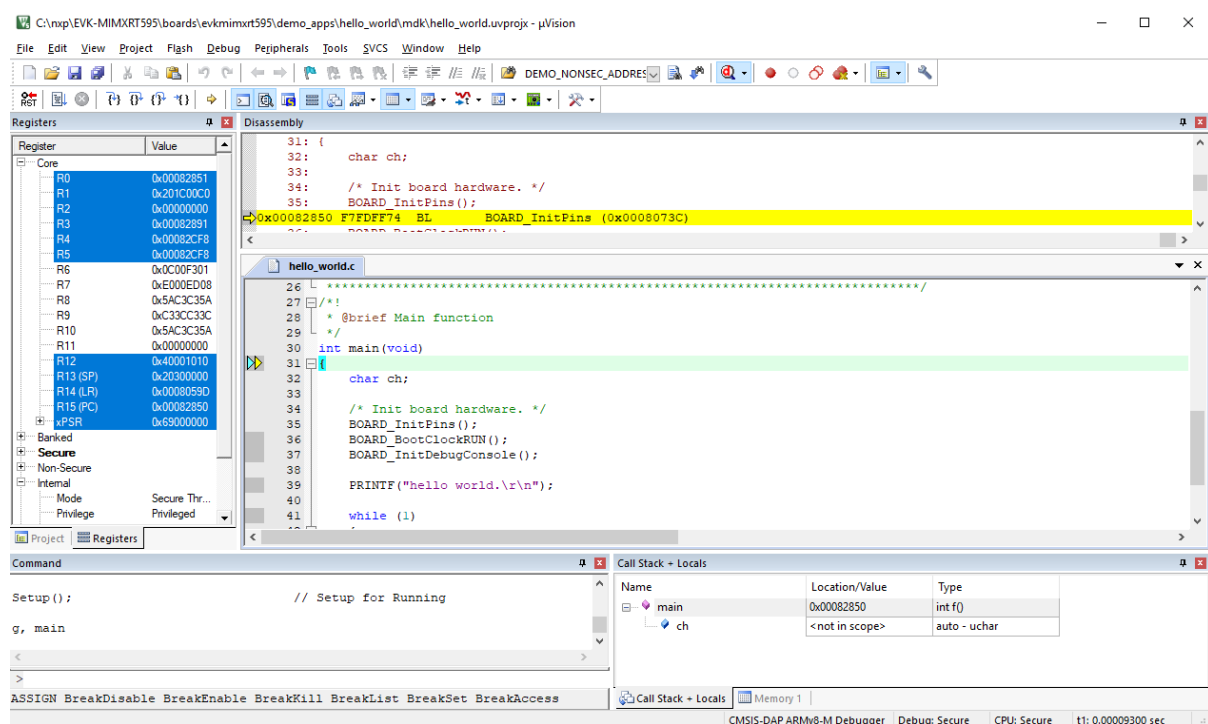
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
→ uvmpw
```

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.

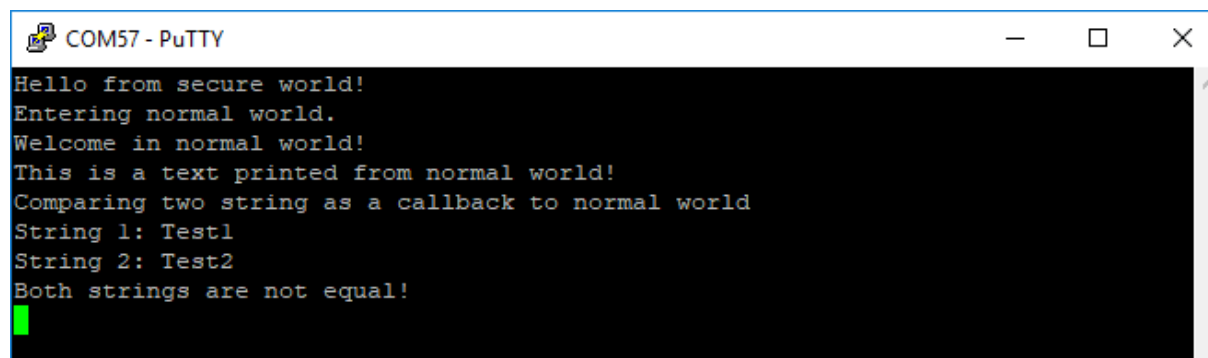
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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 ARMGCC / VSCODE This section describes the steps to run an example application from the SDK archive using the ARMGCC / VSCODE toolchain.

Refer to the [running a demo using MCUXpresso VSC](#) section for detailed instructions on setting up and configuring your project in Visual Studio Code.

Refer to the [CLI](#) section for detailed instructions on building and running your project from the command line.

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 µVision, 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.

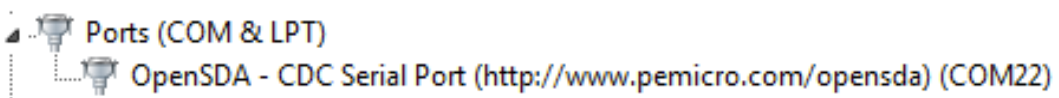
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

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.

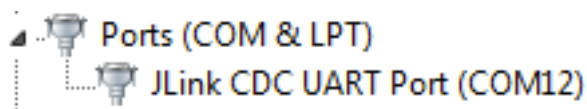
1. **CMSIS-DAP/mbed/DAPLink** interface:



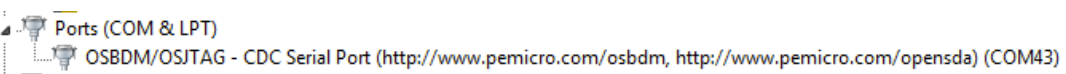
2. **P&E Micro:**



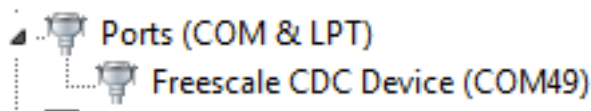
3. **J-Link:**



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.

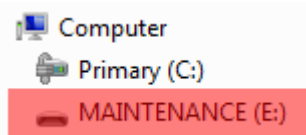
- For boards with CMSIS-DAP firmware, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- For boards with a P&E Micro interface, see [PE micro](#) to download 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-MCXA174	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-MCXA577	CMSIS-DAP	MCU-Link
FRDM-MCXC041	CMSIS-DAP	MCU-Link
FRDM-MCXC162	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

continues on next page

Table 1 – continued from previous page

Hardware platform	Default debugger firmware	On-board debugger probe
FRDM-MCXXN236	CMSIS-DAP	MCU-Link
FRDM-MCXXN947	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
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
}
```

Repository-Layout SDK Package

Development Tools Installation This guide explains how to install the essential tools for development with the MCUXpresso SDK.

Quick Start: Automated Installation (Recommended) The **MCUXpresso Installer** is the fastest way to get started. It automatically installs all the basic tools you need.

1. **Download the MCUXpresso Installer** from: [MCUXpresso Installer](#)
2. **Run the installer** and select “**MCUXpresso SDK Developer**” from the menu
3. **Click Install** and let it handle everything automatically

Manual Installation If you prefer to install tools manually or need specific versions, follow these steps:

Essential Tools

Git - Version Control **What it does:** Manages code versions and downloads SDK repositories from GitHub.

Installation:

- Visit git-scm.com
- Download for your operating system
- Run installer with default settings
- **Important:** Make sure “Add Git to PATH” is selected during installation

Setup:

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

Python - Scripting Environment **What it does:** Runs build scripts and SDK tools.

Installation:

- Install Python **3.10 or newer** from python.org
- **Important:** Check “Add Python to PATH” during installation

West - SDK Management Tool **What it does:** Manages SDK repositories and provides build commands. The west tool is developed by the Zephyr project for managing multiple repositories.

Installation:

```
pip install -U west
```

Minimum version: 1.2.0 or newer

Build System Tools

CMake - Build Configuration **What it does:** Configures how your projects are built.

Recommended version: 3.30.0 or newer

Installation:

- **Windows:** Download .msi installer from cmake.org/download
- **Linux:** Use package manager or download from cmake.org
- **macOS:** Use Homebrew (brew install cmake) or download from cmake.org

Ninja - Fast Build System **What it does:** Compiles your code quickly.

Minimum version: 1.12.1 or newer

Installation:

- **Windows:** Usually included, or download from ninja-build.org
- **Linux:** sudo apt install ninja-build or download binary
- **macOS:** brew install ninja or download binary

Ruby - IDE Project Generation (Optional) **What it does:** Generates project files for IDEs like IAR and Keil.

When needed: Only if you want to use traditional IDEs instead of VS Code.

Installation: Follow the Ruby environment setup guide

Compiler Toolchains Choose and install the compiler toolchain you want to use:

Toolchain	Best For	Download Link	Environment Variable
ARM GCC (Recommended)	Most users, free	ARM Toolchain GNU	ARMGCC_DIR
IAR EWARM	Professional development	IAR Systems	IAR_DIR
Keil MDK	ARM ecosystem	ARM Developer	MDK_DIR
ARM Compiler	Advanced optimization	ARM Developer	ARMCLANG_DIR

Setting Up Environment Variables After toolchain installation, set an environment variable so the build system locates it:

Windows:

```
# Example for ARM GCC installed in C:\armgcc
setx ARMGCC_DIR "C:\armgcc"
```

Linux/macOS:

```
# Add to ~/.bashrc or ~/.zshrc
export ARMGCC_DIR="/usr" # or your installation path
```

Verify Your Installation After installation, verify everything works by opening a terminal/command prompt and running these commands:

```
# Check each tool - you should see version numbers
git --version
python --version
west --version
cmake --version
ninja --version
arm-none-eabi-gcc --version # (if using ARM GCC)
```

Troubleshooting Installation Issues “Command not found” errors:

- The tool isn’t in your system PATH
- **Solution:** Add the installation directory to your PATH environment variable

Python/pip issues:

- Try using python3 and pip3 instead of python and pip
- On Windows, run the Command Prompt as an Administrator

Slow downloads:

- Add timeout option: `pip install -U west --default-timeout=1000`
- Use alternative mirror: `pip install -U west -i https://pypi.tuna.tsinghua.edu.cn/simple`

Building Your First Project This guide explains how to build and run your first SDK example project using the west build system. This applies to both GitHub Repository SDK and Repository-Layout SDK Package.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development board connected via USB
- Build tools installed per [Installation Guide](#)

Understanding Board Support Use the west extension to discover available examples for your board:

```
west list _project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:


```
west list _project -p examples/demo_apps/hello_world -t armgcc
```

Basic Build Process

Simple Build Build the hello_world example with default settings:

```
west build -b your_board examples/demo_apps/hello_world
```

The default toolchain is armgcc, and the build system will select the first debug target as default if no config is specified.

Specifying Configuration

Release build

```
west build -b your_board examples/demo_apps/hello_world --config release
```

Debug build (default)

```
west build -b your_board examples/demo_apps/hello_world --config debug
```

Alternative Toolchains

IAR toolchain

```
west build -b your_board examples/demo_apps/hello_world --toolchain iar
```

Other toolchains as supported by the example

Multicore Applications For multicore devices, specify the core ID:

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

For multicore projects using sysbuild:

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

Flash an Application Flash the built application to your board:

```
west flash -r linkserver
```

Debug Start a debug session:

```
west debug -r linkserver
```

Common Build Options

Clean Build Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run See the commands that get executed without running them:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device DEVICE_PART_NUMBER --config_↵  
↵release
```

Project Configuration

CMake Configuration Only Run configuration without building:

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

Interactive Configuration Launch the configuration GUI:

```
west build -t guiconfig
```

Troubleshooting

Build Failures Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Getting Help View the help information for west build:

```
west build -h
```

Check Supported Configurations To see available configuration options and board targets for an example, refer to the below command:

```
west list_project -p examples/demo_apps/hello_world
```

Next Steps

- Explore other examples in the SDK
- Learn about [Command Line Development](#) for advanced options
- Try [VS Code Development](#) for integrated development
- Refer [Workspace Structure](#) to understand the SDK layout

MCUXpresso for VS Code Development This guide covers using MCUXpresso for VS Code extension to build, debug, and develop SDK applications with an integrated development environment.

Prerequisites

- SDK workspace initialized (GitHub Repository SDK or Repository-Layout SDK Package)
- Development tools installed per [Installation Guide](#)
- Visual Studio Code installed
- MCUXpresso for VS Code extension installed

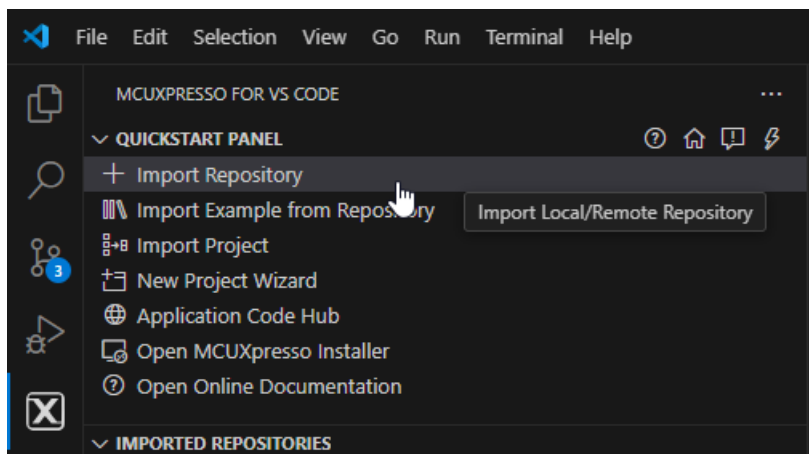
Extension Installation

Install MCUXpresso for VS Code The MCUXpresso for VS Code extension provides integrated development capabilities for MCUXpresso SDK projects. Refer to the [MCUXpresso for VS Code documentation](#) for detailed installation and setup instructions.

SDK Import and Setup

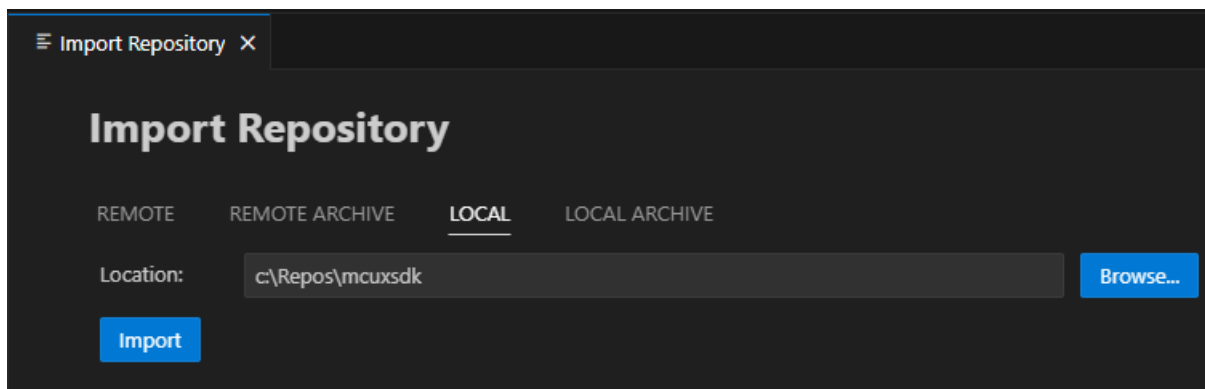
Import Methods The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

Import GitHub Repository SDK Click **Import Repository** from the **QUICKSTART PANEL**

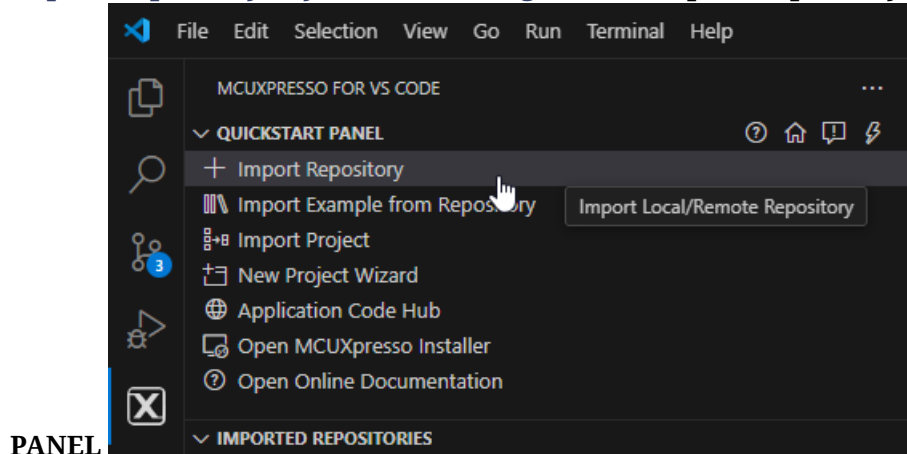


Note: You can import the SDK in several ways. Refer to the [MCUXpresso for VS Code documentation](#) for details.

Select **Local** if you've already obtained the SDK according to [setting up the repo](#). Select your location and click **Import**.

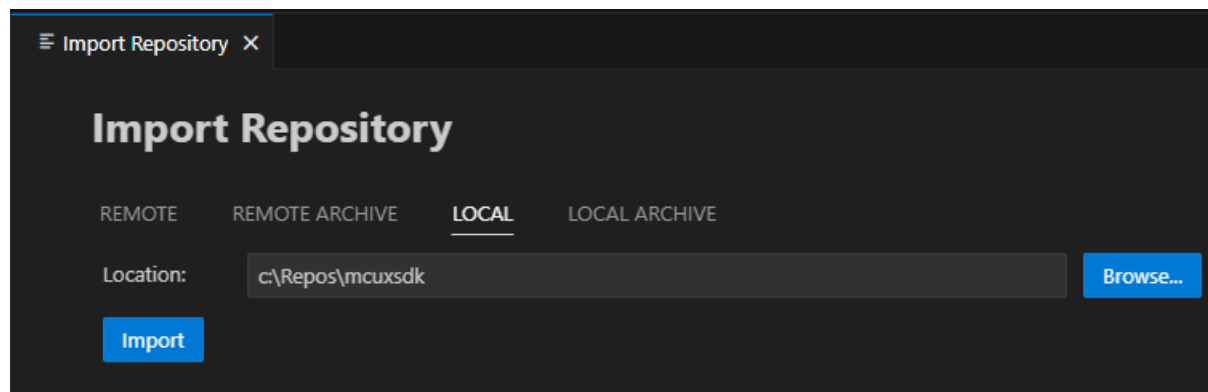


Import Repository-Layout SDK Package Click **Import Repository** from the **QUICKSTART**

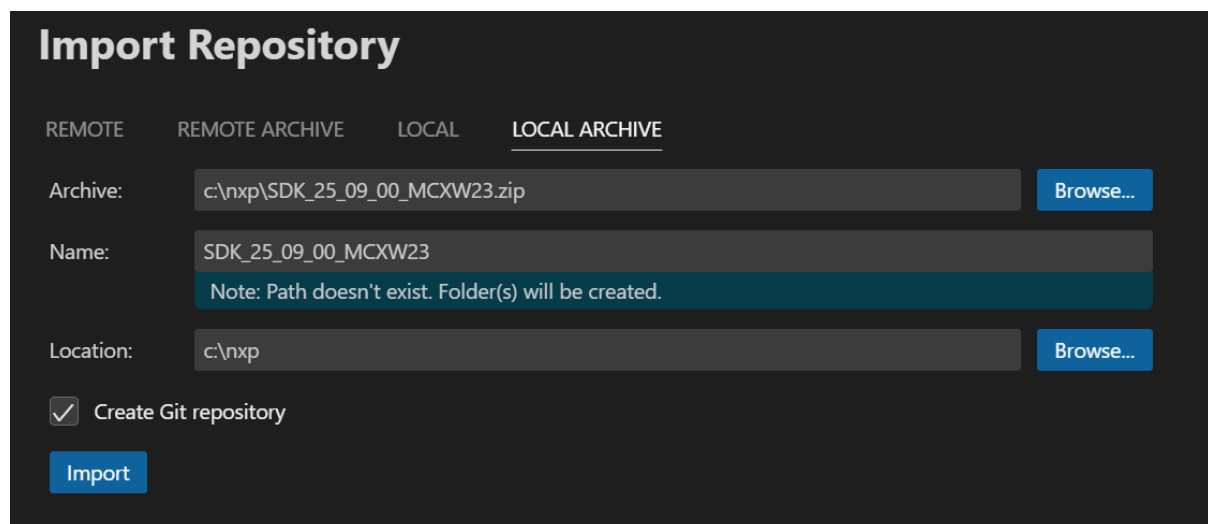


PANEL

Select **Local** if you've already unzipped the Repository-Layout SDK Package. Select your location and click **Import**.



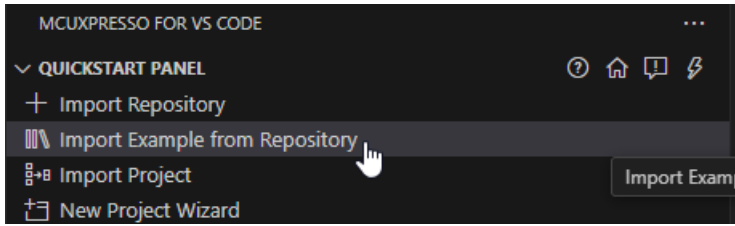
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



Building Example Applications

Import Example Project

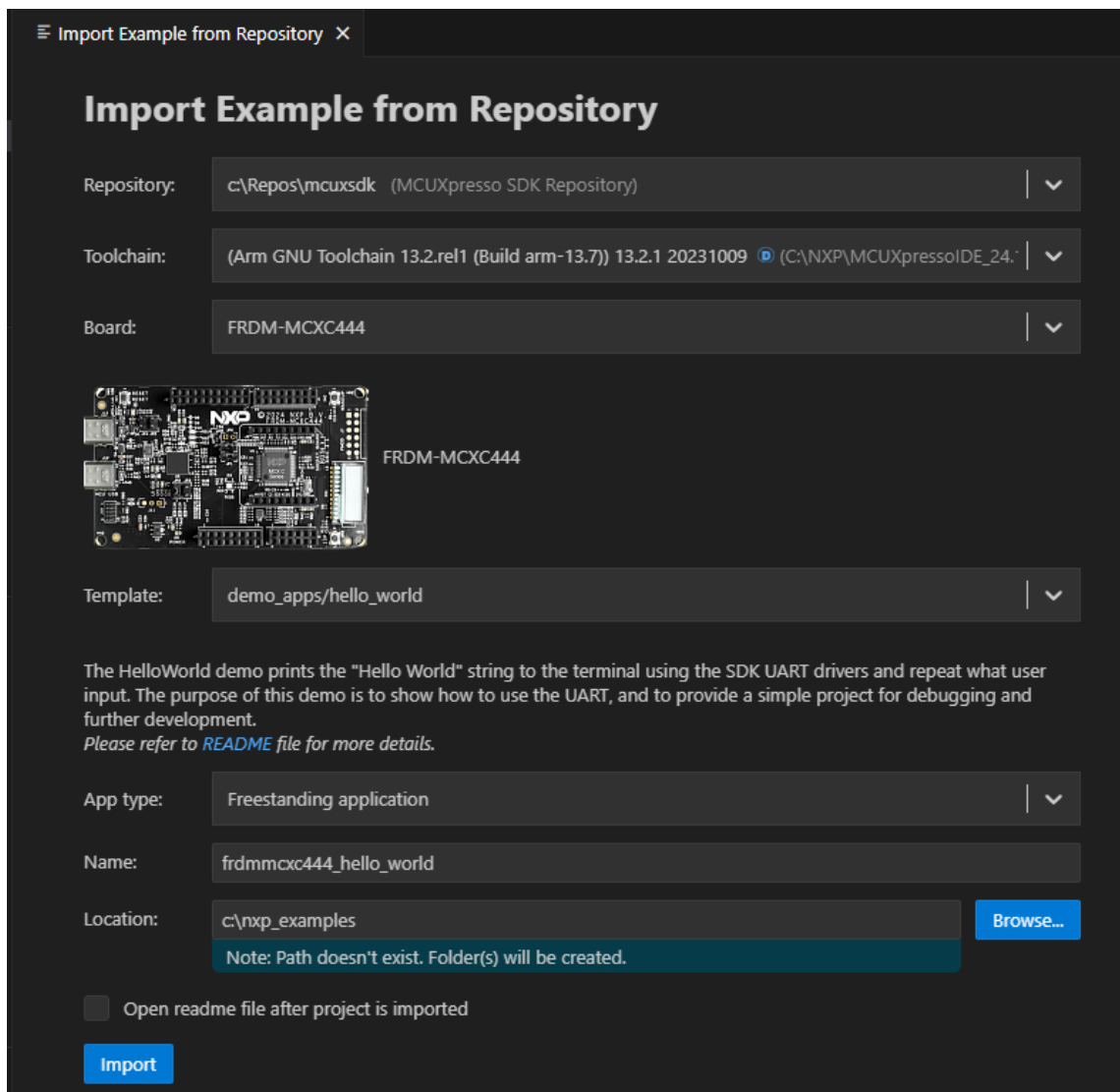
1. Click **Import Example from Repository** from the **QUICKSTART PANEL**



2. Configure project settings:

- **MCUXpresso SDK:** Select your imported SDK
- **Arm GNU Toolchain:** Choose toolchain
- **Board:** Select your target development board
- **Template:** Choose example category
- **Application:** Select specific example (e.g., hello_world)
- **App type:** Choose between Repository applications or Freestanding applications

3. Click **Import**



Application Types Repository Applications:

- Located inside the MCUXpresso SDK
- Integrated with SDK workspace

Freestanding Applications:

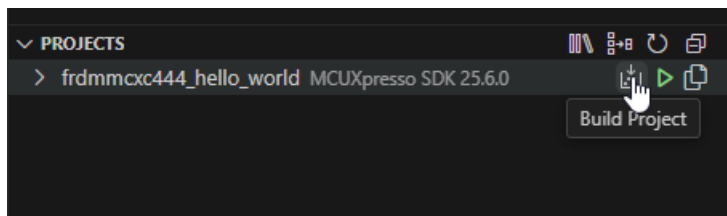
- Imported to user-defined location
- Independent of SDK location

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

Building Projects

Build Process

1. Navigate to **PROJECTS** view
2. Find your project
3. Click the **Build Project** icon

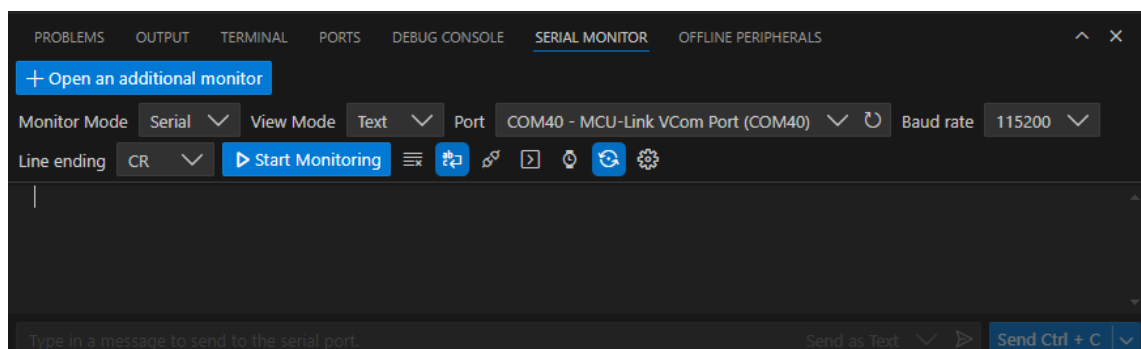


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

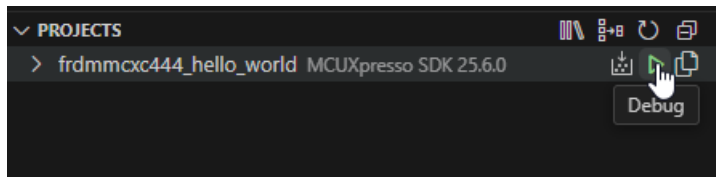
1. Open **Serial Monitor** from VS Code's integrated terminal



2. Configure serial settings:
 - **VCom Port:** Select port for your device
 - **Baud Rate:** Set to 115200

Debug Session

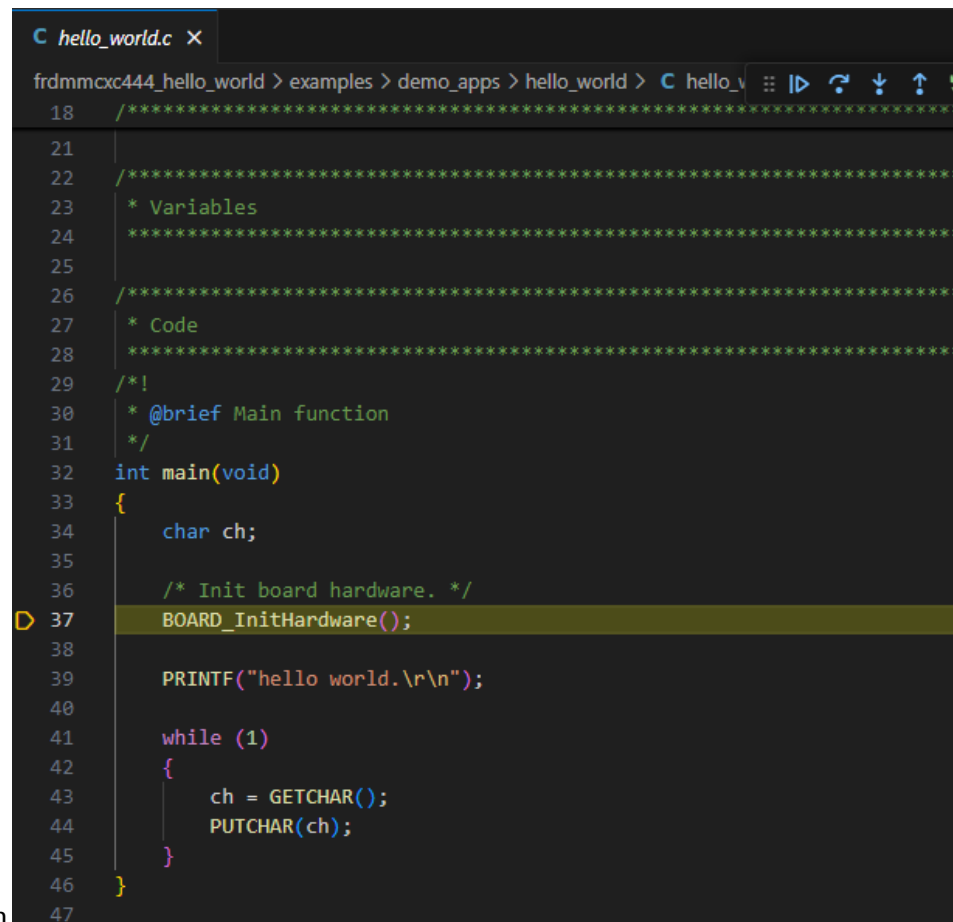
1. Navigate to **PROJECTS** view
2. Click the play button to initiate a debug session



The debug session will begin with debug controls initially at the top of the interface.

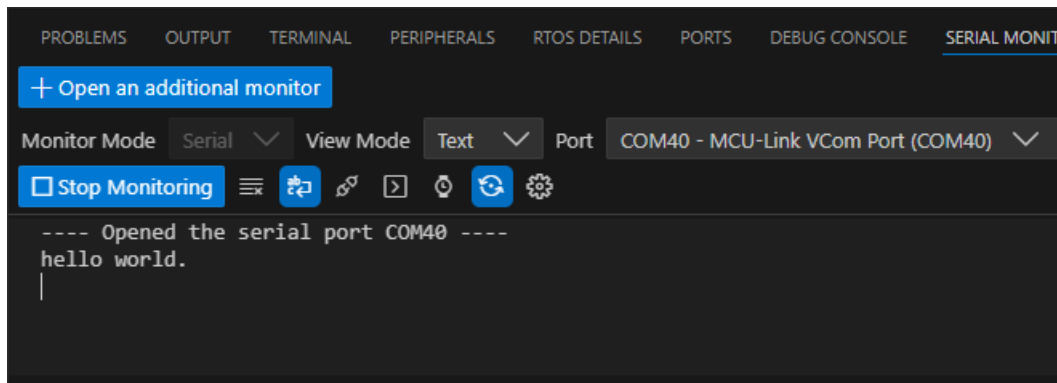
Debug Controls Use the debug controls to manage execution:

- **Continue:** Resume code execution
- **Step controls:** Navigate through code



- **Stop:** Terminate debug session

Monitor Output Observe application output in the **Serial Monitor** to verify correct operation.



Debug Probe Support For comprehensive information on debug probe support and configuration, refer to the [MCUXpresso for VS Code Debug documentation](#).

Project Configuration

Workspace Management The extension integrates with the MCUXpresso SDK workspace structure, providing access to:

- Example applications
- Board configurations
- Middleware components
- Build system integration

Multi-Project Support The PROJECTS view allows management of multiple imported projects within the same workspace.

Troubleshooting

Import Issues SDK not detected:

- Verify SDK workspace is properly initialized
- Ensure all required repositories are updated
- Check SDK manifest files are present

Project import failures:

- Confirm board support exists for selected example
- Verify toolchain installation
- Check example compatibility with selected board

Build Problems Build failures:

- Check integrated terminal for error messages
- Verify all dependencies are installed
- Ensure toolchain is properly configured

Debug Issues **Debug session fails:**

- Verify board connection via USB
- Check debug probe drivers are installed
- Confirm build completed successfully

Serial monitor problems:

- Verify correct VCom port selection
- Check baud rate configuration (115200)
- Ensure board drivers are installed

Integration with Command Line MCUXpresso for VS Code integrates with the underlying west build system, allowing seamless integration with command line workflows described in [Command Line Development](#).

Advanced Features

Project Types The extension supports both repository-based and freestanding project types, providing flexibility in project organization and SDK integration.

Build System Integration The extension leverages the MCUXpresso SDK build system, providing access to all build configurations and options available through command line tools.

Next Steps

- Explore additional examples in the SDK
- Review [Command Line Development](#) for advanced build options
- Refer to the [MCUXpresso for VS Code documentation](#) for detailed documentation
- Learn about [SDK Architecture](#) for better understanding of the development environment

Command Line Development This guide covers developing with the MCUXpresso SDK using command line tools and the west build system. This workflow applies to both GitHub Repository SDK and Repository-Layout SDK Package distributions.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development tools installed per [Installation Guide](#)
- Target board connected via USB

Understanding Board Support Use the west extension to discover available examples for your board:

```
west list _project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

```
west list _project -p examples/demo_apps/hello_world -t armgcc
```

Basic Build Commands

Standard Build Process Build with default settings (armgcc toolchain, first debug config):

```
west build -b your_board examples/demo_apps/hello_world
```

Specifying Build Configuration

Release build

```
west build -b your_board examples/demo_apps/hello_world --config release
```

Debug build with specific toolchain

```
west build -b your_board examples/demo_apps/hello_world --toolchain iar --config debug
```

Multicore Applications For multicore devices, specify the core ID:

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

For multicore projects using sysbuild:

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

Shield Support For boards with shields:

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

Advanced Build Options

Clean Builds Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run See what commands would be executed:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device MK22F12810 --config release
```

Project Configuration

CMake Configuration Only Run configuration without building:

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

Interactive Configuration Launch the configuration GUI:

```
west build -t guiconfig
```

Flashing and Debugging

Flash Application Flash the built application to your board:

```
west flash -r linkserver
```

Debug Session Start a debugging session:

```
west debug -r linkserver
```

IDE Project Generation Generate IDE project files for traditional IDEs:

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

IDE project files are generated in `mcuxsdk/build/<toolchain>` folder.

Note: Ruby installation is required for IDE project generation. See [Installation Guide](#) for setup instructions.

Troubleshooting

Build Failures Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Toolchain Issues Verify environment variables are set correctly:

```
# Check ARM GCC
echo $ARMGCC_DIR
arm-none-eabi-gcc --version

# Check IAR (if using)
echo $IAR_DIR
```

Getting Help Display help information:

```
west build -h
west flash -h
west debug -h
```

Check Supported Configurations If unsure about supported options for an example:

```
west list _project -p examples/demo_apps/hello_world
```

Best Practices

Project Organization

- Keep custom projects outside the SDK tree
- Use version control for your application code
- Document any SDK modifications

Build Efficiency

- Use `-p` always for clean builds when troubleshooting
- Leverage `--dry-run` to understand build processes
- Use specific configs and toolchains to reduce build time

Development Workflow

1. Start with existing examples closest to your requirements
2. Copy and modify rather than building from scratch
3. Test with `hello_world` before moving to complex examples
4. Use configuration tools for pin muxing and clock setup

Next Steps

- Explore [VS Code Development](#) for integrated development experience
- Review [Workspace Structure](#) to understand SDK organization
- Refer build system documentation for advanced configurations

Workspace Structure After you initialize your SDK workspace, it creates a specific directory structure that organizes all SDK components. This structure is identical for both GitHub Repository SDK and Repository-Layout SDK Package.

Top-Level Organization

```
your-sdk-workspace/  
  manifests/      # West manifest repository  
  mcuxsdk/        # Main SDK content
```

The `mcuxsdk/` directory serves as your primary working directory and contains all the SDK components.

SDK Component Layout Based on the actual SDK structure, the main directories include:

Di- rec- tory	Contents	Purpose
arch/	Architecture-specific files	ARM CMSIS, build configurations
cmake/	Build system modules	CMake configuration and build rules
compo	Software components	Reusable software libraries and utilities
devices	Device support packages	MCU-specific headers, startup code, linker scripts
drivers	Peripheral drivers	Hardware abstraction layer for MCU peripherals
examp	Sample applications	Demonstration code and reference implementations
middle	Optional software stacks	Networking, graphics, security, and other libraries
rtos/	Operating system support	FreeRTOS integration
scripts	Build and utility scripts	West extensions and development tools
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.	

Example Organization Examples follow a two-tier structure separating common code from board-specific implementations:

Common Example Files

```
examples/demo_apps/hello_world/
  CMakeLists.txt      # Build configuration
  example.yml         # Example metadata
  hello_world.c       # Application source code
  Kconfig             # Configuration options
  readme.md           # General documentation
```

Board-Specific Files

```
examples/_boards/your_board/demo_apps/hello_world/
  app.h              # Board specific application header
  example_board_readme.md # Board specific documentation
  hardware_init.c    # Board specific hardware initialization
  pin_mux.c          # Pin multiplexing configuration
  pin_mux.h          # Pin multiplexing header definitions
  hello_world.bin     # Pre-built binary for quick testing
  hello_world.mex     # MCUXpresso Config Tools project file
  prj.conf           # Board specific Kconfig configuration
  reconfig.cmake     # Board specific cmake configuration overrides
```

Device Support Structure Device support is organized hierarchically by MCU family:

```
devices/  
  MCX/           # MCU portfolio  
    MCXW/        # MCU family  
      MCXW235/    # Specific device  
        MCXW235.h # Device register definitions  
      drivers/    # Device-specific drivers  
      gcc/        # GNU toolchain files  
      iar/        # IAR toolchain files  
      mcuxpresso/ # MCUXpresso IDE files  
      startup_MCXW235.c # Startup and vector table  
      system_MCXW235.c # System initialization
```

Middleware Organization Middleware components are categorized by functionality and maintained in separate repositories. Based on the manifest files, common middleware categories include:

- **Connectivity:** USB, TCP/IP, industrial protocols
- **Security:** Cryptographic libraries, secure boot
- **Wireless:** Bluetooth, IEEE 802.15.4, Wi-Fi
- **Graphics:** Display drivers, UI frameworks
- **Audio:** Processing libraries, voice recognition
- **Machine Learning:** Inference engines, neural networks
- **Safety:** IEC60730B safety libraries
- **Motor Control:** Motor control and real-time control libraries

Documentation Structure SDK documentation is distributed across multiple locations:

- docs/ - Core SDK documentation and build infrastructure
- Component repositories - API documentation and integration guides
- Board directories - Hardware-specific setup instructions

For complete documentation, refer to the [online documentation](#).

Understanding Example Structure Each example has **two README files**:

1. General README: examples/demo_apps/hello_world/readme.md

- What the example does
- General functionality description
- Common usage information

2. Board-Specific README: examples/_boards/{board_name}/demo_apps/hello_world/example_board_readme.md

- Board-specific setup instructions
- Hardware connections required
- Board-specific behavior notes

Tip: Always check both readme files - start with the general one, then read the board-specific one for detailed setup.

1.3 Getting Started with MCUXpresso SDK GitHub

1.3.1 Getting Started with MCUXpresso SDK Repository

Welcome to the **GitHub Repository SDK Guide**. This documentation provides instructions for setting up and working with the MCUXpresso SDK distributed in a **multi-repository model**. The SDK is distributed across multiple GitHub repositories and managed using the **Zephyr West** tool, enabling modular development and streamlined workflows.

Overview

The GitHub Repository SDK approach offers:

- **Modular Structure:** Multiple repositories for flexibility and scalability.
- **Zephyr West Integration:** Simplified repository management and synchronization.
- **Cross-Platform Support:** Designed for MCUXpresso SDK development environments.

Benefits of the Multi-Repository Approach

- **Scalability:** Easily add or update components without impacting the entire SDK.
- **Collaboration:** Enables distributed development across teams and repositories.
- **Version Control:** Independent versioning for components ensures better stability.
- **Automation:** Zephyr West simplifies dependency handling and repository synchronization.

Setup and Configuration

Follow these steps to prepare your development environment:

GitHub Repository Setup This guide explains how to initialize your MCUXpresso SDK workspace from GitHub repositories using the west tool. The GitHub Repository SDK uses multiple repositories hosted on GitHub to provide modular, flexible development.

Prerequisites Verify the requirements:

System Requirements:

- Python 3.8 or later
- Git 2.25 or later
- CMake 3.20 or later
- Build tools for your target platform

Verification Commands:

```
python --version  # Should show 3.8+
git --version     # Should show 2.25+
cmake --version   # Should show 3.20+
west --version    # Should show west tool installation
```

Workspace Initialization The GitHub Repository SDK uses the Zephyr west tool to manage multiple repositories containing different SDK components.

Step 1: Initialize Workspace Create and initialize your SDK workspace from GitHub:

Get the latest SDK from main branch:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk
```

Get SDK at specific revision:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk --mr {revision}
```

Note: Replace {revision} with the desired release tag, such as v25.09.00

Step 2: Choose Your Repository Update Strategy Navigate to the SDK workspace:

```
cd mcuxpresso-sdk
```

The west tool manages multiple GitHub repositories containing different SDK components. You have two options for downloading:

Option A: Download All Repositories (Complete SDK) Download all SDK repositories for comprehensive development:

```
west update
```

This command downloads all the repositories defined in the manifest from GitHub. Initial download takes several minutes and requires ~7 GB of disk space.

Best for:

- Exploring the complete SDK
- Multi-board development projects
- Comprehensive middleware evaluation

Option B: Targeted Repository Download (Recommended) Download only repositories needed for your specific board or device to save time and disk space:

```
# For specific board development
west update_board --set board your_board_name

# For specific device family development
west update_board --set device your_device_name

# List available repositories before downloading
west update_board --set board your_board_name --list-repo
```

Best for:

- Single board development

- Faster setup and reduced disk usage
- Focused development workflows

Examples:

```
# Update only repositories for FRDM-MCXW23 board
west update_board --set board frdm-mcxw23

# Update only repositories for MCXW23 device family
west update_board --set device mcxw23
```

Step 3: Verify Installation Confirm successful setup:

```
# Verify workspace structure
ls -la
# Should show: manifests/ and mcuxsdk/ directories

# Test build system
west list_project -p examples/demo_apps/hello_world
# Should display available build configurations
```

Advanced Repository Management The west extension command `update_board` provides advanced repository management capabilities for optimized workspace setup with GitHub repositories.

Board-Specific Setup Update only repositories required for a specific board:

```
# Update only repositories for specific board, e.g., frdm-mcxw23
west update_board --set board frdm-mcxw23

# List available repositories for the board before updating
west update_board --set board frdm-mcxw23 --list-repo
```

Device-Specific Setup Update only repositories required for a specific device family:

```
# Update only repositories for specific device, e.g., MCXW235
west update_board --set device mcxw23

# List available repositories for the device family
west update_board --set device mcxw23 --list-repo
```

Custom Configuration For advanced users who want to create custom repository combinations:

```
# Use custom configuration file
west update_board --set custom path/to/custom-config.yml

# Generate custom configuration template
cp manifests/boards/custom.yml.template my-custom-config.yml
```

Benefits of Targeted Setup **Reduced Download Size**

- Download only components needed for your target board or device
- Significantly faster initial setup for focused development

- Typical reduction from 7 GB to 2GB

Optimized Workspace

- Cleaner workspace with relevant components only
- Reduced disk space usage
- Faster repository operations

Flexible Development

- Switch between different board configurations easily
- Maintain separate workspaces for different projects
- Include optional components as needed

Repository Information Before setting up your workspace, you can explore what repositories are available:

```
# Display repository information in console
west update_board --set board frdm-mcxcw23 --list-repo

# Export repository information to YAML file for reference
west update_board --set board frdm-mcxcw23 --list-repo -o board-repos.yml
```

This command lists all the available repositories with descriptions and outlines the included components in the workspace.

Package Generation (Optional) The `update_board` command can also generate ZIP packages for offline distribution:

```
# Generate board-specific SDK package
west update_board --set board frdm-mcxcw23 -o frdm-mcxcw23-sdk.zip
```

Note: Package generation is primarily intended for creating custom SDK distributions. For regular development, use the workspace update commands without the `-o` option.

Workspace Management

Updating Your Workspace Keep your SDK current with latest updates from GitHub:

For Complete SDK Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update all component repositories
cd ..
west update
```

For Targeted Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update board-specific repositories
cd ..
west update_board --set board your_board_name
```

Workspace Status Check workspace synchronization status:

```
# Show status of all repositories
west status
```

```
# Show detailed information about repositories
west list
```

Troubleshooting Network Issues:

- Use `west update --keep-descendants` for partial failures
- Configure Git credentials for private repositories
- Check firewall settings for Git protocol access

Permission Issues:

- Ensure write permissions in workspace directory
- Run commands without `sudo`/administrator privileges
- Verify Git SSH key configuration for authenticated access

Disk Space:

- Full SDK workspace requires approximately 7-8 GB
- Targeted workspace typically requires 1-2 GB
- Use board-specific setup to reduce workspace size

Repository Management Issues:

- Verify board/device names match available configurations
- Check that custom YAML files follow the correct template format
- Use `--list-repo` to verify available repositories before setup

Next Steps With your workspace initialized:

1. Review [Workspace Structure](#) to understand the layout
2. Build your first project with [First Build Guide](#)
3. Explore [Development Workflows MCUXpresso VSCode](#) or [Development Workflows Command Line](#) for the details on project setup and execution

For advanced repository management, see the [west tool documentation](#).

Explore SDK Structure and Content

Learn about the organization of the SDK and its components:

SDK Architecture Overview The MCUXpresso SDK uses a modular architecture where software components are distributed across multiple repositories hosted on GitHub and managed through the west tool. This approach provides flexibility, maintainability, and enables selective component inclusion.

Repository Organization Based on the manifest structure, the SDK consists of four main repository categories:

Manifest Repository The manifest repo (mcuxsdk-manifests) contains the west.yml manifest file that tracks all other repositories in the SDK.

Base Repositories Recorded in submanifests/base.yml and loaded in the root west.yml manifest file. These are the foundational repositories that build the SDK:

- **Devices:** MCU-specific support packages
- **Examples:** Demonstration applications and code samples
- **Boards:** Board support packages

Middleware Repositories Recorded in the submanifests/middleware subdirectory, categorized according to functionality:

- **Connectivity:** Networking stacks, USB, and communication protocols
- **Security:** Cryptographic libraries and secure boot components
- **Wireless:** Bluetooth, IEEE 802.15.4, and other wireless protocols
- **Graphics:** Display drivers and UI frameworks
- **Audio:** Audio processing and voice recognition libraries
- **Machine Learning:** AI inference engines and neural network libraries
- **Safety:** IEC60730B safety libraries
- **Motor Control:** Motor control and real-time control libraries

Internal Repositories Recorded in submanifests/internal.yml and grouped into the “bifrost” group. These are only visible to NXP internal developers and hosted on NXP internal git servers.

Repository Hosting Public repositories are hosted on GitHub under these organizations:

- [nxp-mcuxpresso](#)
- [NXP](#)
- [nxp-zephyr](#)

Internal repositories are hosted on NXP's private Git infrastructure.

Benefits of This Architecture **Selective Integration:** Projects include only required components, reducing memory footprint and build complexity.

Independent Versioning: Each component maintains its own release cycle and version control.

Community Collaboration: Public repositories accept community contributions through standard Git workflows.

Scalable Maintenance: Component owners can update their repositories without affecting the entire SDK.

Workspace Management The west tool manages repository synchronization, version tracking, and workspace updates. All repositories are checked out under the mcuxsdk/ directory with their designated paths defined in the manifest files.

Development Workflows

Get started with building and running projects:

Using MCUXpresso Config Tools MCUXpresso Config tools provide a user-friendly way to configure hardware initialization of your projects. This guide explains the basic workflow with the MCUXpresso SDK west build system and the Config Tools.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- MCUXpresso Config Tools standalone installed (version 25.09 or above)
- MCUXpresso SDK Project that can be successfully built

Board Files MCUXpresso Config Tools generate source files for the board. These files include `pin_mux.c/h` and `clock_config.c/h`. The files contain initialization code functions that reflect the hardware configuration in the Config Tools. Within the SDK codebase, these files are specific for the board and either shared by multiple example projects or specific for one example. Open or import the configuration from the SDK project in the Config Tools and customize the settings to match the custom board or specific project use case and regenerate the code. See *User Guide for MCUXpresso Config Tools (Desktop)* (document [GSMCUXCTUG](#)) for details.

Note: When opening the configuration for SDK example projects, the board files may be shared across multiple examples. To ensure a separate copy of the board configuration files exists, create a freestanding project with copied board files.

Visual Studio Code To open the configuration in Visual Studio Code, use the context menu for the project to access Config Tools. See [MCUXpresso Extension Documentation](#) for details. Otherwise, use the manual workflow described in detail in the following section.

Manual Workflow Use the following steps:

1. Before using Config Tools, run the west command to get the project information for Config Tools from the SDK project files, for example:

```
west cfg_project_info -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_
↪id=cm33_core0
```

This results in the creation of the project information json file that is searched by the config tools when the configuration is created. The parameters of the command should match the build parameters that will be used for the project.

2. Launch the MCUXpresso Config Tools and in the **Start development** wizard, select **Create a new configuration based on the existing IDE/Toolchain project**. Select the created “cfg_tools” subfolder as a project folder (for example: `...mcuxsdk/examples/demo_apps/hello_world/cfg_tools/`).

Updating the SDK West project **Note:** Updating project is supported with Config Tools V25.12 or newer only.

Changes in the Config tools generated source code modules may require adjustments to the toolchain project to ensure a successful build. These changes may mean, for example, adding the newly generated files, adding include paths, required drivers, or other SDK components.

This section describes how to manually resolve the changes needed in the project within the toolchain projects based on the SDK project managed by the West tool.

After the configuration in the Config Tools is finished, write updated files to the disk using the 'Update Code' command. The written files include a json file with the required changes for the toolchain project.

To resolve the changes in the project in the terminal, launch the west command that updates the project. For example:

```
west cfg_resolve -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_id=cm33_core0
```

This command updates the appropriate cmake and kconfig files to address the changes. After this, the application can be built.

Note: The `cfg_resolve` command supports additional arguments. Launch the `west cfg_resolve -h` command to get the list and description.

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 9.70.2
- Keil MDK, version is 5.42a
- MCUXpresso for VS Code v26.03
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

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

Devel- opment boards	MCU devices				
FRDM- MCXC16	MCXC151VFG, MCXC161VFG, MCXC162VFG,	MCXC151VFK, MCXC161VFK, MCXC162VFK,	MCXC151VFM, MCXC161VFM, MCXC162VFM,	MCXC151VFT, MCXC161VFT, MCXC162VFT,	MCXC151VLE, MCXC161VLE, MCXC162VLF

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

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.

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.

1.5 ChangeLog

1.5.1 MCUXpresso SDK Changelog

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

[2.6.3]

- New Features
 - Added bit mask inversion macros to avoid type promotion.
 - Added register operation macros.
- Improvements
 - Make function `MSDK_EnableCpuCycleCounter` compatible with CMSIS-5 and CMSIS-6.
- Bug Fixes
 - Fixed build issue of CMSIS PACK BSP example caused by CMSIS 6.1 issue.

[2.6.2]

- Bug Fixes
 - Fixed violations of MISRA C-2012 rule for implicit conversions in boolean contexts

[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

[2.1.0]

- Bug fix:
 - Choosing CRC clocks from CRC clock array according to instance instead of hardcoded value.

[2.0.5]

- Bug fix:
 - Fix CERT-C issue with boolean-to-unsigned integer conversion.

[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**[2.3.4]**

- Bug Fixes
 - Fixed ERRATA ERR053024 CTIMER will enter interrupt twice when function clock much slower than bus clock.

[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**[2.10.9]**

- Bug Fixes
 - Add new api EDMA_TcdInit to avoid destroying code logic by reordering blocks in the toolchain.

[2.10.8]

- Bug Fixes
 - Fixed coverity issues with CERT INT30-C, CERT INT31-C compliance.
 - Fixed incorrect enabling of preemption capability issue.

[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.
-

EIM

[2.1.0]

- New Features
 - Enhanced EIM driver support for extended memory channels (16-30).
 - Added new API `EIM_InjectDataWordBitError()` and `EIM_GetDataWordBitMask()` to support more data words (2-6).
- Improvements
 - Change `eim_memory_channel_t` to `uint32_t`.

[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.
-

EQDC

[2.3.2]

- Bug Fix
- Fixed violations of the CERT INT31-C.
- Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.8.

[2.3.1]

- Bug Fix
- Fixed CTRL2[CMODE] field overwritten in API EQDC_Init.

[2.3.0]

- Improvements
- Add feature macro to support platforms which do not have compare interrupt.

[2.2.3]

- Bug Fix
- Clear Revolution Counter Register(REV) in init function to prevent its value not equal to zero after reset.

[2.2.2]

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

[2.2.1]

- Bug Fix
- Fixed violations of the MISRA C-2012 rules 20.9.

[2.2.0]

- New features
- Supported the feature that the position counter to be initialized by Index Event Edge Mark.

[2.1.0]

- Bug Fix
- Fixed typo in interrupt enumeration values.
 - Improvements
- Supported Count Direct Change interrupt.
- Removed unused parameter in user configuration.
- Supported ERRATA_051383 check, the CTRL[DMAEN] can't be cleared.

[2.0.1]

- Bug Fix
- Fixed violations of the MISRA C-2012 rules 10.3, 10.6, 10.8, 14.4, 16.4.

[2.0.0]

- Initial version.

ERM

[2.0.4]

- New Features
 - Enhanced ERM driver support for extended memory channels (16-30).
- Improvements
 - Change `erm_memory_channel_t` to `uint32_t`.

[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.
-

FLASH

[2.3.3]

- Improvements
 - Added timeout for while loop in `flash_command_pre_sequence`, `flash_command_sequence`, `flash_erase_sequence`, `flash_pgm_sequence` and `flash_command_pre_sequence` API.

[2.3.2]

- Improvement
 - Enabled the `kFLASH_PropertyPflash1SectorSize` property support in `FLASH_GetProperty` API.

[2.3.1]

- Improvement
- Clear flash cache before every erase to prevent the possibility of returning stale data.

[2.3.0]

- New features
- Add support for MCX N series

[2.2.1]

- Improvement
 - To fix the potential unsigned integer overflow in the operation, added a check to ensure that the addition does not exceed the maximum value.

[2.2.0]

- New Features
 - Add the support for MW30.

[2.1.2]

- Improvement
 - Add the support for KW47/MCXW72 phantoms.

[2.1.1]

- Improvement
 - Add the conditional compiling flag ‘#if defined(RF_FMU)’ to make the driver be compatible with the non-radio phantoms.

[2.1.0]

- Bug Fixes
 - Fix flash driver run error in flash memory

[2.0.0]

- Initial version.
-

FREQME**[2.1.4]**

- Improvements
 - Clarified pulse width measurement mode usage.

[2.1.3]

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

[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.
-

GPIO

[2.8.4]

- Improvements
 - Make function `GPIO_PortGetInterruptFlags` and `GPIO_PortClearInterruptFlags` available for all variants, they are used for all pins flags in one GPIO port.

[2.8.3]

- Bug Fixes
 - Fixed violations of the MISRA C-2012 Rule 10.1, 5.7.

[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_GetPinsDMAResourceFlags 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_CONTROL.

[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`”.
-

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.
-

LPADC

[2.9.5]

- Improvements
 - Fix doxygen issue, grouping command should be balanced.

[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.
-

LPI2C

[2.6.4]

- Bug Fixes
 - Limited value of filtSda in LPI2C_MasterSetBaudRate().
 - Updated LPI2C_MasterStop() and LPI2C_MasterTransferBlocking() to send I2C STOP also in case of error.
 - Removed unused flag kLPI2C_TransferRepeatedStartFlag.

[2.6.3]

- Bug Fixes
 - Fixed static analysis identified issues.

[2.6.2]

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

[2.6.1]

- Bug Fixes
 - Fixed coverity issues.

[2.6.0]

- New Feature
 - Added common IRQ handler entry LPI2C_DriverIRQHandler.

[2.5.7]

- Improvements
 - Added support for separated IRQ handlers.

[2.5.6]

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

[2.5.5]

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

[2.5.4]

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

[2.5.3]

- Improvements
 - Added handler for LPI2C7 and LPI2C8.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

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

[2.3.0]

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

[2.2.0]

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

[2.1.12]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.1.11]

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

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

[2.1.10]

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

LPI2C_EDMA

[2.4.7]

- Bug Fixes
 - Fixed incorrect TX FIFO size (maxTxFifo) in LPI2C_MasterTransferEDMA() and LPI2C_MasterTransferEdmaHandleIRQ().

[2.4.6]

- Bug Fixes
 - Fixed static analysis identified issues.

[2.4.5]

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

[2.4.4]

- Improvements
 - Added support for 2KB data transfer

[2.4.3]

- Improvements
 - Added support for separated IRQ handlers.

[2.4.2]

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

[2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1
-

LPSPI**[2.7.4]**

- Bug Fixes
 - Clear WIDTH bits from the TCR register before writing a new value in LP-SPI_MasterTransferBlocking().

[2.7.3]

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

[2.7.2]

- Bug Fixes
 - Fixed coverity issues.

[2.7.1]

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

[2.7.0]

- New Feature
 - Added common IRQ handler entry LPSPI_DriverIRQHandler.

[2.6.10]

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

[2.6.9]

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

[2.6.8]

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

[2.6.7]

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

[2.6.6]

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

[2.6.5]

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

[2.6.4]

- Bug Fixes
 - Added LPSPI6_DriverIRQHandler for LPSPI6 instance.

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

- Improvements
 - Improved LPSPI_SlaveTransferHandleIRQ to fill up TX FIFO instead of write one data to TX register which improves the slave transmit performance.
 - Added new functional APIs LPSPI_SelectTransferPCS and LPSPI_SetPCSContinuous to support changing PCS selection and PCS continuous mode.
- Bug Fixes
 - Fixed bug in non-blocking and EDMA transfer APIs that kStatus_InvalidArgument is returned if user configures 3-wire mode and full-duplex transfer at the same time, but transfer state is already set to kLPSPI_Busy by mistake causing following transfer can not start.
 - Fixed bug when LPSPI slave using EDMA way to transfer, tx should be masked when tx data is null, otherwise in 3-wire mode which tx/rx use the same pin, the received data will be interfered.

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the bug that the clock source should be divided by the PRESCALE setting in LPSPI_MasterSetDelayTimes function.
 - Fixed the bug that LPSPI_MasterTransferBlocking function would hang in some corner cases.
- Optimization
 - Added #ifndef/#endif to allow user to change the default TX value at compile time.

[2.0.0]

- Initial version.
-

LPSPI_EDMA

[2.4.9]

- Improvements
 - Removed unused code from LPSPI_SeparateEdmaReadData().

[2.4.8]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

- Improvements
 - Add the TXMSK wait after TCR setting.

[2.4.0]

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

LPTMR**[2.2.1]**

- Bug Fixes
 - Fix CERT INT31-C issues.

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

LPUART

[2.11.0]

- New Feature
 - Added support to enable transmitter RTS and configure transmitter RTS polarity.
 - Added support to enable Transmit Data Inversion.
- Improvements
 - Updated to work on DSC platform.

[2.10.0]

- New Feature
 - Added support to configure RTS watermark.

[2.9.4]

- Improvements
 - Merged duplicate code.

[2.9.3]

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

[2.9.2]

- Bug Fixes
 - Fixed coverity issues.

[2.9.1]

- Bug Fixes
 - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.7]

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

[2.7.6]

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

[2.7.5]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

- Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
 - Added APIs LPUART_GetRxFifoCount/LPUART_GetTxFifoCount to get rx/tx FIFO data count.
 - Added APIs LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
 - Fixed MISRA advisory 17.7 issues.

[2.4.0]

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

[2.3.1]

- Bug Fixes
 - Fixed MISRA advisory 15.5 issues.

[2.3.0]

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

[2.2.8]

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

- Modified LPUART_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
 - Update transactional APIs.
-

LPUART_EDMA

[2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0
-

OPAMP

[2.2.1]

- Improvements
 - Fixed CERT-C issues.

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

PORT**[2.5.1]**

- Bug Fixes
 - Fix CERT INT31-C issues.
 - Fixed the violations of MISRA C-2012 rules: 10.1.

[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.
-

PWM

[2.9.4]

- Bug Fixes
 - Fixed CERT INT31-C issues.

[2.9.3]

- Improvements
 - Cherry-pick patch from hal_nxp repository. Enhanced input validation in PWM_SetupPwm by returning kStatus_InvalidArgument when the calculated pulse count exceeds UINT16_MAX, to prevent silent failures when assertions are disabled in production builds.

[2.9.2]

- Improvements
 - Add new API `PWM_GetInputCaptureValue` to read the value captured from the submodule counter.

[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.
-

RESET

[2.0.0]

- Initial version.
-

RTC

[2.4.0]

- New features
 - Add support for RTC clock output.
 - Add support for RTC time seconds interrupt configuration.

[2.3.3]

- Bug Fixes
 - Fix RTC_GetDatetime function validating datetime issue.

[2.3.2]

- Improvements
 - Handle errata 010716: Disable the counter before setting alarm register and then reen-able the counter.

[2.3.1]

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.3.0]

- Improvements
 - Added API RTC_EnableLPOClock to set 1kHz LPO clock.
 - Added API RTC_EnableCrystalClock to replace API RTC_SetClockSource.

[2.2.2]

- Improvements
 - Refine _rtc_interrupt_enable order.

[2.2.1]

- Bug Fixes
 - Fixed the issue of Pa082 warning.
 - Fixed the issue of bit field mask checking.
 - Fixed the issue of hard code in RTC_Init.

[2.2.0]

- Bug Fixes
 - Fixed MISRA C-2012 issue.
 - * Fixed rule contain: rule-17.7, rule-14.4, rule-10.4, rule-10.7, rule-10.1, rule-10.3.
 - Fixed central repository code formatting issue.
- Improvements
 - Added an API for enabling wakeup pin.

[2.1.0]

- Improvements
 - Added feature macro check for many features.

[2.0.0]

- Initial version.
-

MCX_SPC**[2.12.0]**

- New Features
 - Add kSPC_controlVbat and kSPC_controlTsi0 to spc_analog_module_control_t for TSI0 and VBAT control support.

[2.11.0]

- New Features
 - Add feature macro `FSL_FEATURE_MCX_SPC_HAS_SRAMRETLDO_REFTRIM_REG` to be compatible with some platforms where SPC has no `SRAMRETLDO_REFTRIM` register.

[2.10.1]

- Improvements
 - Add support for some platform does not has Trim for `SRAMRETLDO`.

[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.
-

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.
-

TRNG**[2.0.21]**

- New features:
 - Added support for MCXC devices.

[2.0.20]

- New features:
 - Added support for MCXA devices.

[2.0.19]

- New features:
 - Added support for MCXA and MCXL.

[2.0.18]

- Bug fix:
 - TRNG health checks now done in software on RT5xx and RT6xx.

[2.0.17]

- New features:
 - Add support for RT700.

[2.0.16]

- Improvements:
 - Added support for Dual oscillator mode.

[2.0.15]

- Other changes:
 - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest recommended by design team.

[2.0.14]

- New features:
 - Add support for RW610 and RW612.

[2.0.13]

- Bug fix:
 - After deepsleep it might return error, added clearing bits in TRNG_GetRandomData() and generating new entropy.
 - Modified reloading entropy in TRNG_GetRandomData(), for some data length it doesn't reloading entropy correctly.

[2.0.12]

- Bug fix:
 - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.

[2.0.11]

- Bug fix:
 - Add clearing pending errors in TRNG_Init().

[2.0.10]

- Bug Fix:
 - Fixed doxygen issues.

[2.0.9]

- Bug Fix:
 - Fix HIS_CCM metrics issues.

[2.0.8]

- Bug fix:
 - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.

[2.0.7]

- Bug fix:
 - Fix MISRA 2004 issue rule 12.5.

[2.0.6]

- Bug fix:
 - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.

[2.0.5]

- Improvements:
 - For FRQMIN, FRQMAX and OSCDIV, add possibility to use device specific preprocessor macro to define default value in TRNG user configuration structure.

[2.0.4]

- Bug Fix:
 - Fix MISRA-2012 issues.
 - * Rule 10.1, rule 10.3, rule 13.5, rule 16.1.

[2.0.3]

- Improvements:
 - update TRNG_Init to restart new entropy generation.

[2.0.2]

- Improvements:
 - fix MISRA issues
 - * Rule 14.4.

[2.0.1]

- New features:
 - Set default OSCDIV for Kinetis devices KL8x and KL28Z.
- Other changes:
 - Changed default OSCDIV for K81 to divide by 2.

[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.
-

WUU

[2.4.2]

- Bug Fixes
 - Fixed violations of CERT-C:
 - * Check filterIndex >= 1U to prevent underflow.
 - * Check multiplication results fit in uint8_t before assignment to shift variable.

[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.

[MCXC162](#)

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 MCU Boot

[mcuboot_opensource](#)

1.7.2 FreeMASTER

[freemaster](#)

1.7.3 FreeRTOS

[FreeRTOS](#)

1.7.4 File systemFatfs

[FatFs](#)

Chapter 2

MCXC162

2.1 Clock Driver

enum _clock_ip_name

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

enumerator kCLOCK__InputMux

Clock gate name: INPUTMUX0

enumerator kCLOCK__GateINPUTMUX0

Clock gate name: INPUTMUX0

enumerator kCLOCK__GateFREQME

Clock gate name: FREQME

enumerator kCLOCK__GateCTIMER0

Clock gate name: CTIMER0

enumerator kCLOCK__GateCTIMER1

Clock gate name: CTIMER1

enumerator kCLOCK__GateWWDT0

Clock gate name: WWDT0

enumerator kCLOCK__GateWWDT1

Clock gate name: WWDT1

enumerator kCLOCK__GateDMA0

Clock gate name: DMA0

enumerator kCLOCK__GateCRC0

Clock gate name: CRC0

enumerator kCLOCK__Crc0

Clock gate name: CRC0

enumerator kCLOCK__GateEIM0

Clock gate name: EIM0

enumerator kCLOCK__GateERM0

Clock gate name: ERM0

enumerator kCLOCK_GateFLEXPWM0

 Clock gate name: FLEXPWM0

enumerator kCLOCK_GateLPI2C0

 Clock gate name: LPI2C0

enumerator kCLOCK_GateLPSPI0

 Clock gate name: LPSPI0

enumerator kCLOCK_GateLPUART0

 Clock gate name: LPUART0

enumerator kCLOCK_GateLPUART1

 Clock gate name: LPUART1

enumerator kCLOCK_GateLPUART2

 Clock gate name: LPUART2

enumerator kCLOCK_GateLPUART3

 Clock gate name: LPUART3

enumerator kCLOCK_GateADC0

 Clock gate name: ADC0

enumerator kCLOCK_GateCMP0

 Clock gate name: CMP0

enumerator kCLOCK_GateOPAMP0

 Clock gate name: OPAMP0

enumerator kCLOCK_GatePORT0

 Clock gate name: PORT0

enumerator kCLOCK_GatePORT1

 Clock gate name: PORT1

enumerator kCLOCK_GatePORT2

 Clock gate name: PORT2

enumerator kCLOCK_GatePORT3

 Clock gate name: PORT3

enumerator kCLOCK_GateGPIO0

 Clock gate name: GPIO0

enumerator kCLOCK_GateGPIO1

 Clock gate name: GPIO1

enumerator kCLOCK_GateGPIO2

 Clock gate name: GPIO2

enumerator kCLOCK_GateGPIO3

 Clock gate name: GPIO3

enumerator kCLOCK_GateRAMA

 Clock gate name: RAMA

enumerator kCLOCK_GateTRNG0

 Clock gate name: TRNG0

enumerator kCLOCK_GateSECCON

 Clock gate name: SECCON

enumerator kCLOCK_GatePWM0SM0

Clock gate name: PWM0 SM0

enumerator kCLOCK_GatePWM0SM1

Clock gate name: PWM0 SM1

enumerator kCLOCK_GatePWM0SM2

Clock gate name: PWM0 SM2

enumerator kCLOCK_GateNotAvail

Clock gate name: None

enum _clock_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_MainClk

MAIN_CLK

enumerator kCLOCK_CoreSysClk

Core/system clock(CPU_CLK)

enumerator kCLOCK_SYSTEM_CLK

SYSTEM clock/AHB_BUS

enumerator kCLOCK_BusClk

SYSTEM clock

enumerator kCLOCK_ExtClk

External Clock

enumerator kCLOCK_FroHf

FRO192

enumerator kCLOCK_FroHfDiv

Divided by FRO192

enumerator kCLOCK_Fro12M

FRO12M

enumerator kCLOCK_Fro12MDiv

FRO12MDiv

enumerator kCLOCK_Clk1M

CLK1M

enumerator kCLOCK_Fro16K

FRO16K

enumerator kCLOCK_Clk16K0

CLK16K[0]

enumerator kCLOCK_Clk16K1

CLK16K[1]

enumerator kCLOCK_Osc32K0

OSC32K[0]

enumerator kCLOCK_Osc32K1

OSC32K[1]

enumerator kCLOCK_SLOW_CLK
SYSTEM_CLK divided by 4

enum _clock_select_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_SelCTIMER0
CTIMER0 clock selection

enumerator kCLOCK_SelCTIMER1
CTIMER1 clock selection

enumerator kCLOCK_SelWWDT1
WWDT1 clock selection

enumerator kCLOCK_SelLPI2C0
LPI2C0 clock selection

enumerator kCLOCK_SelLPSPi0
LPSPi0 clock selection

enumerator kCLOCK_SelLPUART0
LPUART0 clock selection

enumerator kCLOCK_SelLPUART1
LPUART1 clock selection

enumerator kCLOCK_SelLPUART2
LPUART2 clock selection

enumerator kCLOCK_SelLPUART3
LPUART3 clock selection

enumerator kCLOCK_SelLPTMR0
LPTMR0 clock selection

enumerator kCLOCK_SelADC
ADC clock selection

enumerator kCLOCK_SelCMP0
CMP0 clock selection

enumerator kCLOCK_SelTRACE
TRACE clock selection

enumerator kCLOCK_SelCLKOUT
CLKOUT clock selection

enumerator kCLOCK_SelSYSTICK
SYSTICK clock selection

enumerator kCLOCK_SelSCGSCS
SCG SCS clock selection

enumerator kCLOCK_SelMax
MAX clock selection

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 kCLK_32K_to_MAIN_CLK
Attach CLK_32K[1] to MAIN_CLK.

enumerator kNONE_to_MAIN_CLK
Attach NONE to MAIN_CLK.

enumerator kFRO_LF_DIV_to_CTIMER0
Attach FRO_LF_DIV to CTIMER0.

enumerator kFRO_HF_to_CTIMER0
Attach FRO_HF to CTIMER0.

enumerator kCLK_IN_to_CTIMER0
Attach CLK_IN to CTIMER0.

enumerator kCLK_16K_to_CTIMER0
Attach CLK_16K to CTIMER0.

enumerator kCLK_1M_to_CTIMER0
Attach CLK_1M to CTIMER0.

enumerator kNONE_to_CTIMER0
Attach NONE to CTIMER0.

enumerator kFRO_LF_DIV_to_CTIMER1
Attach FRO_LF_DIV to CTIMER1.

enumerator kFRO_HF_to_CTIMER1
Attach FRO_HF to CTIMER1.

enumerator kCLK_IN_to_CTIMER1
Attach CLK_IN to CTIMER1.

enumerator kCLK_16K_to_CTIMER1
Attach CLK_16K to CTIMER1.

enumerator kCLK_1M_to_CTIMER1
Attach CLK_1M to CTIMER1.

enumerator kNONE_to_CTIMER1
Attach NONE to CTIMER1.

enumerator kFRO_LF_DIV_to_LPI2C0
Attach FRO_LF_DIV to LPI2C0.

enumerator kFRO_HF_DIV_to_LPI2C0
Attach FRO_HF_DIV to LPI2C0.

enumerator kCLK_IN_to_LPI2C0
Attach CLK_IN to LPI2C0.

enumerator kCLK_1M_to_LPI2C0
Attach CLK_1M to LPI2C0.

enumerator kNONE_to_LPI2C0
Attach NONE to LPI2C0.

enumerator kFRO_LF_DIV_to_LPSPI0
Attach FRO_LF_DIV to LPSPI0.

enumerator kFRO_HF_DIV_to_LPSPI0
Attach FRO_HF_DIV to LPSPI0.

enumerator kCLK_IN_to_LPSPI0
Attach CLK_IN to LPSPI0.

enumerator kCLK_1M_to_LPSPI0
Attach CLK_1M to LPSPI0.

enumerator kNONE_to_LPSPI0
Attach NONE to LPSPI0.

enumerator kFRO_LF_DIV_to_LPUART0
Attach FRO_LF_DIV to LPUART0.

enumerator kFRO_HF_DIV_to_LPUART0
Attach FRO_HF_DIV to LPUART0.

enumerator kCLK_IN_to_LPUART0
Attach CLK_IN to LPUART0.

enumerator kCLK_32K_to_LPUART0
Attach CLK_16K to LPUART0.

enumerator kCLK_1M_to_LPUART0
Attach CLK_1M to LPUART0.

enumerator kNONE_to_LPUART0
Attach NONE to LPUART0.

enumerator kFRO_LF_DIV_to_LPUART1
Attach FRO_LF_DIV to LPUART1.

enumerator kFRO_HF_DIV_to_LPUART1
Attach FRO_HF_DIV to LPUART1.

enumerator kCLK_IN_to_LPUART1
Attach CLK_IN to LPUART1.

enumerator kCLK_32K_to_LPUART1
Attach CLK_16K to LPUART1.

enumerator kCLK_1M_to_LPUART1
Attach CLK_1M to LPUART1.

enumerator kNONE_to_LPUART1
Attach NONE to LPUART1.

enumerator kFRO_LF_DIV_to_LPUART2
Attach FRO_LF_DIV to LPUART2.

enumerator kFRO_HF_DIV_to_LPUART2
Attach FRO_HF_DIV to LPUART2.

enumerator kCLK_IN_to_LPUART2
Attach CLK_IN to LPUART2.

enumerator kCLK_32K_to_LPUART2
Attach CLK_16K to LPUART2.

enumerator kCLK_1M_to_LPUART2
Attach CLK_1M to LPUART2.

enumerator kNONE_to_LPUART2
Attach NONE to LPUART2.

enumerator kFRO_LF_DIV_to_LPUART3
Attach FRO_LF_DIV to LPUART2.

enumerator kFRO_HF_DIV_to_LPUART3
Attach FRO_HF_DIV to LPUART2.

enumerator kCLK_IN_to_LPUART3
Attach CLK_IN to LPUART2.

enumerator kCLK_32K_to_LPUART3
Attach CLK_16K to LPUART2.

enumerator kCLK_1M_to_LPUART3
Attach CLK_1M to LPUART2.

enumerator kNONE_to_LPUART3
Attach NONE to LPUART2.

enumerator kFRO_LF_DIV_to_LPTMR0
Attach FRO_LF_DIV to LPTMR0.

enumerator kFRO_HF_DIV_to_LPTMR0
Attach FRO_HF_DIV to LPTMR0.

enumerator kCLK_IN_to_LPTMR0
Attach CLK_IN to LPTMR0.

enumerator kCLK_1M_to_LPTMR0
Attach CLK_1M to LPTMR0.

enumerator kNONE_to_LPTMR0
Attach NONE to LPTMR0.

enumerator kFRO_LF_DIV_to_ADC
Attach FRO_LF_DIV to ADC.

enumerator kFRO_HF_to_ADC
Attach FRO_HF to ADC.

enumerator kCLK_IN_to_ADC
Attach CLK_IN to ADC.

enumerator kCLK_1M_to_ADC
Attach CLK_1M to ADC.

enumerator kNONE_to_ADC
Attach NONE to ADC.

enumerator kFRO_LF_DIV_to_CMP0
Attach FRO_LF_DIV to CMP0.

enumerator kFRO_HF_DIV_to_CMP0
Attach FRO_HF_DIV to CMP0.

enumerator kCLK_IN_to_CMP0
Attach CLK_IN to CMP0.

enumerator kCLK_1M_to_CMP0
Attach CLK_1M to CMP0.

enumerator kNONE_to_CMP0
Attach NONE to CMP0.

enumerator kCPU_CLK_to_TRACE
Attach CPU_CLK to TRACE.

enumerator kCLK_1M_to_TRACE
Attach CLK_1M to TRACE.

enumerator kCLK_16K_to_TRACE
Attach CLK_16K to TRACE.

enumerator kNONE_to_TRACE
Attach NONE to TRACE.

enumerator kFRO12M_to_CLKOUT
Attach FRO_12M to CLKOUT.

enumerator kFRO_HF_DIV_to_CLKOUT
Attach FRO_HF_DIV to CLKOUT.

enumerator kCLK_IN_to_CLKOUT
Attach CLK_IN to CLKOUT.

enumerator kCLK_16K_to_CLKOUT
Attach CLK_16K to CLKOUT.

enumerator kSLOW_CLK_to_CLKOUT
Attach SLOW_CLK to CLKOUT.

enumerator kNONE_to_CLKOUT
Attach NONE to CLKOUT.

enumerator kCPU_CLK_to_SYSTICK
Attach CPU_CLK to SYSTICK.

enumerator kCLK_1M_to_SYSTICK
Attach CLK_1M to SYSTICK.

enumerator kCLK_16K_to_SYSTICK
Attach CLK_16K to SYSTICK.

enumerator kNONE_to_SYSTICK
Attach NONE to SYSTICK.

enumerator kCLK_16K_to_WWDT1
Attach CLK_16K to WWDT1.

enumerator kFRO_HF_DIV_to_WWDT1
Attach FRO_HF_DIV to WWDT1.

enumerator kCLK_1M_to_WWDT1
Attach CLK_1M to WWDT1.

enumerator kNONE_to_WWDT1
Attach NONE to WWDT1.

enumerator kNONE_to_NONE

Attach NONE to NONE.

enum _clock_div_name

Clock dividers.

Values:

enumerator kCLOCK_DivAHBCLK

AHBCLK clock divider

enumerator kCLOCK_DivSLOWCLK

SLOWCLK clock divider

enumerator kCLOCK_DivFRO_HF

FROHF clock divider

enumerator kCLOCK_DivFRO_LF

FROLF clock divider

enumerator kCLOCK_DivCTIMER0

CTIMER0 clock divider

enumerator kCLOCK_DivCTIMER1

CTIMER1 clock divider

enumerator kCLOCK_DivWWDT0

WWDT0 clock divider

enumerator kCLOCK_DivWWDT1

WWDT1 clock divider

enumerator kCLOCK_DivLPI2C0

LPI2C0 clock divider

enumerator kCLOCK_DivLPSPI0

LPSPI0 clock divider

enumerator kCLOCK_DivLPUART0

LPUART0 clock divider

enumerator kCLOCK_DivLPUART1

LPUART1 clock divider

enumerator kCLOCK_DivLPUART2

LPUART2 clock divider

enumerator kCLOCK_DivLPUART3

LPUART3 clock divider

enumerator kCLOCK_DivLPTMR0

LPTMR0 clock divider

enumerator kCLOCK_DivADC

ADC clock divider

enumerator kCLOCK_DivCMP0_FUNC

CMP0_FUNC clock divider

enumerator kCLOCK_DivCMP0_RR

CMP0_RR clock divider

enumerator kCLOCK_DivTRACE
TRACE clock divider

enumerator kCLOCK_DivCLKOUT
CLKOUT clock divider

enumerator kCLOCK_DivSYSTICK
SYSTICK clock divider

enumerator kCLOCK_DivMax
MAX clock divider

enum _clk16k_clk_gate_id
CLK16K clock gate.

Values:

enumerator kCLKE_16K_SYSTEM
CLK16K[0] to SYSTEM domain.

enumerator kCLKE_16K_COREMAIN
CLK16K[1] to CORE domain.

enumerator kCLOCK_Clk16kToSys
CLK16K[0] to SYSTEM domain.

enumerator kCLOCK_Clk16kToCore
CLK16K[1] to CORE domain.

enumerator kCLOCK_Clk16kToSysAndCore
CLK16K to SYSTEM and CORE domain.

enum _scg_status
SCG status return codes.

Values:

enumerator kStatus_SCG_Busy
Clock is busy.

enumerator kStatus_SCG_InvalidSrc
Invalid source.

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 sirc_trim_mode_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 kNoTrimSrc
No external trim source.

enumerator kSCG_SircTrimSrcSysOsc
System OSC.

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 _run_mode
The active run mode (voltage level).

Values:

enumerator kSD_Mode
SD Mode

enum _osc32k_clk_gate_id
OSC32K clock gate.

Values:

enumerator kCLOCK_Osc32kToSys
OSC32K[0] to SYSTEM domain.

enumerator kCLOCK_Osc32kToCore
OSC32K[1] to CORE domain.

enumerator kCLOCK_Osc32kToSysAndCore
OSC32K to SYSTEM and CORE domain.

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 _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`

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_select_name` `clock_select_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 `_clk16k_clk_gate_id` `clk16k_clk_gate_id`
CLK16K clock gate.

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

typedef enum `_osc32k_clk_gate_id` `osc32k_clk_gate_id_t`
OSC32K clock gate.

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 `_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.

```
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

```
void CLOCK_AttachClk(clock_attach_id_t connection)
```

Configure the clock selection muxes.

Parameters

- *connection* – : Clock to be configured.

```
clock_attach_id_t CLOCK_GetClockAttachId(clock_attach_id_t connection)
```

Get the actual clock attach id. This fuction 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

- *connection* – : Clock attach id to get.

Returns

Clock source value.

```
void CLOCK_SetClockSelect(clock_select_name_t sel_name, uint32_t value)
```

Set the clock select value. This fuction set the peripheral clock select value.

Parameters

- *sel_name* – : Clock select.
- *value* – : value to be set.

```
uint32_t CLOCK_GetClockSelect(clock_select_name_t sel_name)
```

Get the clock select value. This fuction get the peripheral clock select value.

Parameters

- *sel_name* – : Clock select.

Returns

Clock source value.

`void CLOCK_SetClockDiv(clock_div_name_t div_name, uint32_t value)`

Setup peripheral clock dividers.

Parameters

- `div_name` – : Clock divider name
- `value` – : Value to be divided

`uint32_t CLOCK_GetClockDiv(clock_div_name_t div_name)`

Get peripheral clock dividers.

Parameters

- `div_name` – : Clock divider name

Returns

peripheral clock dividers

`void CLOCK_HaltClockDiv(clock_div_name_t div_name)`

Halt peripheral clock dividers.

Parameters

- `div_name` – : Clock divider name

`status_t CLOCK_SetupFROHFClocking(uint32_t iFreq)`

Initialize the FROHF to given frequency (48,64,96,192). This function turns on FIRC and select the given frequency as the source of `fro_hf`.

Parameters

- `iFreq` – : Desired frequency.

Returns

returns success or fail status.

`status_t CLOCK_SetupFRO12MClocking(void)`

Initialize the FRO12M. This function turns on FRO12M.

Returns

returns success or fail status.

`status_t CLOCK_SetupFRO16KClocking(uint8_t clk_16k_enable_mask)`

Initialize the FRO16K. This function turns on FRO16K.

Parameters

- `clk_16k_enable_mask` – 0-3 0b00: disable both `clk_16k0` and `clk_16k1` 0b01: only enable `clk_16k0` 0b10: only enable `clk_16k1` 0b11: enable both `clk_16k0` and `clk_16k1`

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.

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_GetLpi2cClkFreq(uint32_t id)

Return Frequency of LPI2C0 functional Clock.

Returns

Frequency of LPI2C0 functional Clock

uint32_t CLOCK_GetLpspiClkFreq(uint32_t id)

Return Frequency of LPSPI functional Clock.

Returns

Frequency of LPSPI functional Clock

uint32_t CLOCK_GetLpuartClkFreq(uint32_t id)

Return Frequency of LPUART functional Clock.

Returns

Frequency of LPUART functional Clock

uint32_t CLOCK_GetLptmrClkFreq(void)

Return Frequency of LPTMR functional Clock.

Returns

Frequency of LPTMR functional Clock

uint32_t CLOCK_GetAdcClkFreq(uint32_t id)

Return Frequency of Adc Clock.

Returns

Frequency of Adc.

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_GetTraceClkFreq(void)

Return Frequency of Trace Clock.

Returns

Frequency of Trace.

uint32_t CLOCK_GetClkoutClkFreq(void)

Return Frequency of CLKOUT Clock.

Returns

Frequency of CLKOUT.

uint32_t CLOCK_GetSystickClkFreq(void)

Return Frequency of Systick Clock.

Returns

Frequency of Systick.

uint32_t CLOCK_GetWwdtClkFreq(void)

brief Return Frequency of Systick Clock return Frequency of Systick.

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.

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

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 calulation and accelerate the execution to get the right settings.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

CLK_GATE_REG_OFFSET(value)

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

CLK_GATE_BIT_SHIFT(value)

REG_PWM0SUBCTL

CRC_CLOCKS

Clock ip name array for CRC.

CTIMER_CLOCKS

Clock ip name array for CTIMER.

DMA_CLOCKS

Clock ip name array for DMA.

EDMA_CLOCKS

Clock gate name array for EDMA.

ERM_CLOCKS

Clock ip name array for ERM.

EIM_CLOCKS

Clock ip name array for EIM.

FREQME_CLOCKS

Clock ip name array for FREQME.

GPIO_CLOCKS

Clock ip name array for GPIO.

INPUTMUX_CLOCKS

Clock ip name array for INPUTMUX.

LPCMP_CLOCKS

Clock ip name array for LPCMP.

LPADC_CLOCKS

Clock ip name array for LPADC.

LPUART_CLOCKS

Clock ip name array for LPUART.

LPI2C_CLOCKS

Clock ip name array for LPI2C.

LPSPI_CLOCKS

Clock ip name array for LPSPI.

OPAMP_CLOCKS

Clock ip name array for OPAMP.

PORT_CLOCKS

Clock ip name array for PORT.

TRNG_CLOCKS

Clock ip name array for TRNG.

WWDT_CLOCKS

Clock ip name array for WWDT.

FLEXPWM_CLOCKS

Clock ip name array for FLEXPWM.

RAMA_CLOCKS

Clock ip name array for RAMA.

SECCON_CLOCKS

Clock ip name array for SECCON.

ATX_CLOCKS

Clock ip name array for ATX.

TCU_CLOCKS

Clock ip name array for TCU.

PWM_CLOCKS

Clock ip name array for PWM.

BUS_CLK

Peripherals clock source definition.

CLK_ATTACH_REG_OFFSET(value)

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]*

CLK_ATTACH_CLK_SEL(value)**CLK_ATTACH_MUX(reg, sel)**

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

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.

2.2 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.1.0.

Current version: 2.1.0

Change log:

- Version 2.1.0
 - Choosing CRC clocks from CRC clock array according to instance instead of hard-coded value.
- Version 2.0.5
 - Fix CERT-C issue with boolean-to-unsigned integer conversion.
- 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.

```
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()

2.3 CTIMER: Standard counter/timers

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

Ungates 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 2.3.4

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], kC-`
`TIMER_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

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

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

Initializes the eDMA peripheral.

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

Note: This function enables the minor loop map feature.

Parameters

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

void `EDMA_Deinit(EDMA_Type *base)`

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- `base` – eDMA peripheral base address.

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

Push content of TCD structure into hardware TCD register.

Parameters

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

void EDMA_GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

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

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

Parameters

- config – A pointer to the eDMA configuration structure.

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

EDMA Channel initialization.

Parameters

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

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

Set channel memory attribute.

Parameters

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

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

Set channel sign extension.

Parameters

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

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

Set channel swap size.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.

- `swapSize` – Swap occurs with respect to the specified transfer size. If 0, swap is disabled.

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

Set channel access type.

Parameters

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

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

Set channel request source.

Parameters

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

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

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

Parameters

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

Returns

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

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

Set channel master ID replication.

Parameters

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

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

Set channel security level.

Parameters

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

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

Sets all TCD registers to default values.

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

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Configures the eDMA transfer attribute.

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

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

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

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

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

Configures the eDMA minor offset feature.

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

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_SetChannelPreemptionConfig(EDMA_Type *base, uint32_t channel, const
                                     edma_channel_preemption_config_t *config)
```

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

```
void EDMA_SetChannelLink(EDMA_Type *base, uint32_t channel, edma_channel_link_type_t
                        type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
 - kEDMA_LinkNone
 - kEDMA_MinorLink
 - kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
void EDMA_SetBandWidth(EDMA_Type *base, uint32_t channel, edma_bandwidth_t
                      bandWidth)
```

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
 - kEDMABandwidthStallNone
 - kEDMABandwidthStall4Cycle
 - kEDMABandwidthStall8Cycle

```
void EDMA_SetModulo(EDMA_Type *base, uint32_t channel, edma_modulo_t srcModulo,  
                  edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

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

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

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

```
void EDMA_EnableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)
```

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined *edma_interrupt_enable_t* type.

```
void EDMA_DisableChannelInterrupts(EDMA_Type *base, uint32_t channel, uint32_t mask)
```

Disables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined *edma_interrupt_enable_t* type.

```
void EDMA_SetMajorOffsetConfig(EDMA_Type *base, uint32_t channel, int32_t sourceOffset,  
                              int32_t destOffset)
```

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- channel – edma channel number.
- sourceOffset – source address offset will be applied to source address after major loop done.

- `destOffset` – destination address offset will be applied to source address after major loop done.

```
void EDMA__ConfigChannelSoftwareTCD(edma_tcd_t *tcd, const edma_transfer_config_t
                                     *transfer)
```

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__ConfigChannelSoftwareTCDExt`

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- `tcd` – Pointer to the TCD structure.
- `transfer` – channel transfer configuration pointer.

```
void EDMA__TcdReset(edma_tcd_t *tcd)
```

Sets all fields to default values for the TCD structure.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdResetExt`

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- `tcd` – Pointer to the TCD structure.

```
void EDMA__TcdSetTransferConfig(edma_tcd_t *tcd, const edma_transfer_config_t *config,
                               edma_tcd_t *nextTcd)
```

Configures the eDMA TCD transfer attribute.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetTransferConfigExt`

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA__TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

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

Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA__TcdSetMinorOffsetConfig(edma_tcd_t *tcd, const edma_minor_offset_config_t *config)
```

Configures the eDMA TCD minor offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetMinorOffsetConfigExt

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

```
void EDMA__TcdSetChannelLink(edma_tcd_t *tcd, edma_channel_link_type_t type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetChannelLinkExt

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
 - kEDMA_LinkNone
 - kEDMA_MinorLink
 - kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
static inline void EDMA__TcdSetBandWidth(edma_tcd_t *tcd, edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API EDMA_TcdSetBandWidthExt

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after

the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- `tcd` – A pointer to the TCD structure.
- `bandWidth` – A bandwidth setting, which can be one of the following:
 - `kEDMABandwidthStallNone`
 - `kEDMABandwidthStall4Cycle`
 - `kEDMABandwidthStall8Cycle`

```
void EDMA__TcdSetModulo(edma_tcd_t *tcd, edma_modulo_t srcModulo, edma_modulo_t
                        destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetModuloExt`

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- `tcd` – A pointer to the TCD structure.
- `srcModulo` – A source modulo value.
- `destModulo` – A destination modulo value.

```
static inline void EDMA__TcdEnableAutoStopRequest(edma_tcd_t *tcd, bool enable)
```

Sets the auto stop request for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdEnableAutoStopRequestExt`

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- `tcd` – A pointer to the TCD structure.
- `enable` – The command to enable (true) or disable (false).

```
void EDMA__TcdEnableInterrupts(edma_tcd_t *tcd, uint32_t mask)
```

Enables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdEnableInterruptsExt`

Parameters

- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

```
void EDMA__TcdDisableInterrupts(edma_tcd_t *tcd, uint32_t mask)
```

Disables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdDisableInterruptsExt`

Parameters

- `tcd` – Point to the TCD structure.

- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__TcdSetMajorOffsetConfig(edma_tcd_t *tcd, int32_t sourceOffset, int32_t destOffset)`

Configures the eDMA TCD major offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API `EDMA__TcdSetMajorOffsetConfigExt`

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- `tcd` – A point to the TCD structure.
- `sourceOffset` – source address offset will be applied to source address after major loop done.
- `destOffset` – destination address offset will be applied to source address after major loop done.

`void EDMA__ConfigChannelSoftwareTCDExt(EDMA_Type *base, edma_tcd_t *tcd, const edma_transfer_config_t *transfer)`

Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.
- `transfer` – channel transfer configuration pointer.

`void EDMA__TcdResetExt(EDMA_Type *base, edma_tcd_t *tcd)`

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Pointer to the TCD structure.

`void EDMA__TcdSetTransferConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const edma_transfer_config_t *config, edma_tcd_t *nextTcd)`

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:


```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

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

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

```
void EDMA_TcdSetMinorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, const
                                     edma_minor_offset_config_t *config)
```

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

```
void EDMA_TcdSetChannelLinkExt(EDMA_Type *base, edma_tcd_t *tcd,
                              edma_channel_link_type_t type, uint32_t linkedChannel)
```

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
 - kEDMA_LinkNone
 - kEDMA_MinorLink

- kEDMA_MajorLink
- linkedChannel – The linked channel number.

```
static inline void EDMA__TcdSetBandWidthExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                             edma_bandwidth_t bandWidth)
```

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- bandWidth – A bandwidth setting, which can be one of the following:
 - kEDMABandwidthStallNone
 - kEDMABandwidthStall4Cycle
 - kEDMABandwidthStall8Cycle

```
void EDMA__TcdSetModuloExt(EDMA_Type *base, edma_tcd_t *tcd, edma_modulo_t srcModulo,  
                          edma_modulo_t destModulo)
```

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

```
static inline void EDMA__TcdEnableAutoStopRequestExt(EDMA_Type *base, edma_tcd_t *tcd,  
                                                    bool enable)
```

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

```
void EDMA__TcdEnableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)
```

Enables the interrupt source for the eDMA TCD.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined *edma_interrupt_enable_t* type.

`void EDMA__TcdDisableInterruptsExt(EDMA_Type *base, edma_tcd_t *tcd, uint32_t mask)`

Disables the interrupt source for the eDMA TCD.

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – Point to the TCD structure.
- `mask` – The mask of interrupt source to be set. Users need to use the defined `edma_interrupt_enable_t` type.

`void EDMA__TcdSetMajorOffsetConfigExt(EDMA_Type *base, edma_tcd_t *tcd, int32_t sourceOffset, int32_t destOffset)`

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- `base` – eDMA peripheral base address.
- `tcd` – A point to the TCD structure.
- `sourceOffset` – source address offset will be applied to source address after major loop done.
- `destOffset` – destination address offset will be applied to source address after major loop done.

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

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

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

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

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

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

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

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

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

`uint32_t EDMA__GetRemainingMajorLoopCount(EDMA_Type *base, uint32_t channel)`

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

- a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: $\text{RemainingBytes} = \text{RemainingMajorLoopCount} * \text{NBYTES}(\text{initially configured})$

Parameters

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

Returns

Major loop count which has not been transferred yet for the current TCD.

```
static inline uint32_t EDMA_GetErrorStatusFlags(EDMA_Type *base)
```

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

The mask of error status flags. Users need to use the `_edma_error_status_flags` type to decode the return variables.

```
uint32_t EDMA_GetChannelStatusFlags(EDMA_Type *base, uint32_t channel)
```

Gets the eDMA channel status flags.

Parameters

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

Returns

The mask of channel status flags. Users need to use the `_edma_channel_status_flags` type to decode the return variables.

```
void EDMA_ClearChannelStatusFlags(EDMA_Type *base, uint32_t channel, uint32_t mask)
```

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined `_edma_channel_status_flags` type.

```
status_t EDMA_CreateHandle(edma_handle_t *handle, EDMA_Type *base, uint32_t channel)
```

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

Return values

- kStatus_Success –
- kStatus_InvalidArgument –

void EDMA_InstallTCDMemory(*edma_handle_t* *handle, *edma_tcd_t* *tcdPool, uint32_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the EDMA_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA_SubmitTransfer.

Parameters

- handle – eDMA handle pointer.
- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

void EDMA_SetCallback(*edma_handle_t* *handle, *edma_callback* callback, void *userData)

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.
- userData – A parameter for the callback function.

void EDMA_PrepareTransferConfig(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE). User can check if 128 bytes support is available for specific instance by FSL_FEATURE_EDMA_INSTANCE_SUPPORT_128_BYTES_TRANSFERn.

Parameters

- config – The user configuration structure of type *edma_transfer_t*.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.

- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.

```
void EDMA__PrepareTransfer(edma_transfer_config_t *config, void *srcAddr, uint32_t srcWidth,  
                          void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest,  
                          uint32_t transferBytes, edma_transfer_type_t type)
```

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- `config` – The user configuration structure of type `edma_transfer_t`.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `type` – eDMA transfer type.

```
void EDMA__PrepareTransferTCD(edma_handle_t *handle, edma_tcd_t *tcd, void *srcAddr,  
                             uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t  
                             destWidth, int16_t destOffset, uint32_t bytesEachRequest,  
                             uint32_t transferBytes, edma_tcd_t *nextTcd)
```

Prepares the eDMA transfer content descriptor.

This function prepares the transfer content descriptor structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- `handle` – eDMA handle pointer.
- `tcd` – Pointer to eDMA transfer content descriptor structure.
- `srcAddr` – eDMA transfer source address.
- `srcWidth` – eDMA transfer source address width(bytes).
- `srcOffset` – source address offset.
- `destAddr` – eDMA transfer destination address.
- `destWidth` – eDMA transfer destination address width(bytes).

- `destOffset` – destination address offset.
- `bytesEachRequest` – eDMA transfer bytes per channel request.
- `transferBytes` – eDMA transfer bytes to be transferred.
- `nextTcd` – eDMA transfer linked TCD address.

`status_t` EDMA_SubmitTransferTCD(*edma_handle_t* *handle, *edma_tcd_t* *tcd)

Submits the eDMA transfer content descriptor.

This function submits the eDMA transfer request according to the transfer content descriptor. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

Typical user case:

a. submit single transfer

```
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

b. submit static link transfer,

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ...)
EDMA_PrepareTransferTCD(handle, &tcd[1], ...)
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

c. submit dynamic link transfer

```
edma_tcd_t tcdpool[2];
EDMA_InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_PrepareTransferTCD(handle, tcd, ...)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

d. submit loop transfer

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ..., &tcd[1])
EDMA_PrepareTransferTCD(handle, &tcd[1], ..., &tcd[0])
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

Parameters

- `handle` – eDMA handle pointer.
- `tcd` – Pointer to eDMA transfer content descriptor structure.

Return values

- `kStatus_EDMA_Success` – It means submit transfer request succeed.
- `kStatus_EDMA_QueueFull` – It means TCD queue is full. Submit transfer request is not allowed.
- `kStatus_EDMA_Busy` – It means the given channel is busy, need to submit request later.

`status_t` EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCMemory 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.

void EDMA_TcdInit(*EDMA_Type* *base, *edma_tcd_t* *tcdRegs)

Initialize all fields to 0 for the TCD structure.

This function initialize all fields for this TCD structure to 0.

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.10.9.

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull

TCD queue is full.

enumerator kStatus_EDMA_Busy

Channel is busy and can't handle the transfer request.

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes

Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA_TransferSize128Bytes

Source/Destination data transfer size is 128 bytes every time

enum _edma_modulo

eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable
Disable modulo

enumerator kEDMA_Modulo2bytes
Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes
Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes
Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes
Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes
Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes

Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes

Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes

Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes

Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes

Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes

Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes

Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes

Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes

Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes

Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes

Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes

Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes

Circular buffer size is 2 G bytes.

enum _edma_bandwidth

Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone

No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle

eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle

eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type

Channel link type.

Values:

enumerator kEDMA_LinkNone

No channel link

enumerator kEDMA_MinorLink

Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.

Values:

enumerator kEDMA_DoneFlag
DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag
eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag
eDMA interrupt flag, set while an interrupt occurred of this channel

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag
Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

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

_edma_interrupt_enable eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type
eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

enum edma_channel_memory_attribute
eDMA channel memory attribute

Values:

enumerator kEDMA_ChannelNoWriteNoReadNoCacheNoBuffer
No write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadNoCacheBufferable
No write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableNoBuffer
No write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableBufferable
No write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheNoBuffer
No write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheBufferable
No write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableNoBuffer
No write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableBufferable
No write allocate, read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheNoBuffer
write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheBufferable
write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableNoBuffer
write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableBufferable
write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheNoBuffer
write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheBufferable
write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadCacheableNoBuffer
write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadCacheableBufferable
write allocate, read allocate, cacheable, bufferable.

enum _edma_channel_swap_size
eDMA4 channel swap size

Values:

enumerator kEDMA_ChannelSwapDisabled
Swap is disabled.

enumerator kEDMA_ChannelReadWith8bitSwap
Swap occurs with respect to the read 8bit.

enumerator kEDMA_ChannelReadWith16bitSwap
Swap occurs with respect to the read 16bit.

enumerator kEDMA_ChannelReadWith32bitSwap
Swap occurs with respect to the read 32bit.

enumerator kEDMA_ChannelWriteWith8bitSwap
Swap occurs with respect to the write 8bit.

enumerator kEDMA_ChannelWriteWith16bitSwap
Swap occurs with respect to the write 16bit.

enumerator kEDMA_ChannelWriteWith32bitSwap
Swap occurs with respect to the write 32bit.

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

enumerator kEDMA_PrivilegedAccessLevel
Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA_MasterId
DMA's master ID when channel is active and master ID replication is enabled.

enum _edma_channel_access_type
eDMA4 channel access type

Values:

enumerator kEDMA_ChannelDataAccess
Data access for eDMA4 transfers.

enumerator kEDMA_ChannelInstructionAccess
Instruction access for eDMA4 transfers.

enum _edma_channel_protection_level
eDMA4 channel protection level

Values:

enumerator `kEDMA_ChannelProtectionLevelUser`
user protection level for eDMA transfers.

enumerator `kEDMA_ChannelProtectionLevelPrivileged`
Privileged protection level eDMA transfers.

typedef enum *_edma_transfer_size* `edma_transfer_size_t`
eDMA transfer configuration

typedef enum *_edma_modulo* `edma_modulo_t`
eDMA modulo configuration

typedef enum *_edma_bandwidth* `edma_bandwidth_t`
Bandwidth control.

typedef enum *_edma_channel_link_type* `edma_channel_link_type_t`
Channel link type.

typedef enum *_edma_transfer_type* `edma_transfer_type_t`
eDMA transfer type

typedef struct *_edma_channel_Preemption_config* `edma_channel_Preemption_config_t`
eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* `edma_minor_offset_config_t`
eDMA minor offset configuration

typedef enum *edma_channel_memory_attribute* `edma_channel_memory_attribute_t`
eDMA channel memory attribute

typedef enum *_edma_channel_swap_size* `edma_channel_swap_size_t`
eDMA4 channel swap size

typedef enum *_edma_channel_access_type* `edma_channel_access_type_t`
eDMA4 channel access type

typedef enum *_edma_channel_protection_level* `edma_channel_protection_level_t`
eDMA4 channel protection level

typedef struct *_edma_channel_config* `edma_channel_config_t`
eDMA4 channel configuration

typedef *edma_core_tcd_t* `edma_tcd_t`
eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

typedef struct *_edma_transfer_config* `edma_transfer_config_t`
edma4 channel transfer configuration

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as

each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

Note: User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
 - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
 - c. the totalBytes should align with the bytesEachRequest
 - d. the srcAddr should align with the srcWidthOfEachTransfer
 - e. the dstAddr should align with the dstWidthOfEachTransfer
 - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
 - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
-

```
typedef struct _edma_config edma_config_t
```

edMA global configuration structure.

```
typedef void (*edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)
```

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Param handle

EDMA handle pointer, users shall not touch the values inside.

Param userData

The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

Param transferDone

If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

Param tcds

How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

```
typedef struct _edma_handle edma_handle_t
```

edMA transfer handle structure

FSL_EDMA_DRIVER_EDMA4

eDMA driver name

EDMA_ALLOCATE_TCD(name, number)

Macro used for allocate edma TCD.

DMA_DCHPRI_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct _edma_channel_Preemption_config

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption

If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility

If true: a channel can suspend other channel with low priority

uint8_t channelPriority

Channel priority

struct _edma_minor_offset_config

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

bool enableSrcMinorOffset

Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset

Enable(true) or Disable(false) destination minor loop offset.

uint32_t minorOffset

Offset for a minor loop mapping.

struct _edma_channel_config

#include <fsl_edma.h> eDMA4 channel configuration

Public Members

edma_channel_Preemption_config_t channelPreemptionConfig

channel preemption configuration

edma_channel_memory_attribute_t channelReadMemoryAttribute

channel memory read attribute configuration

edma_channel_memory_attribute_t channelWriteMemoryAttribute

channel memory write attribute configuration

edma_channel_swap_size_t channelSwapSize

channel swap size configuration

edma_channel_access_type_t channelAccessType

channel access type configuration

uint8_t channelDataSignExtensionBitPosition

channel data sign extension bit position configuration

```
uint32_t channelRequestSource
    hardware service request source for the channel
bool enableMasterIDReplication
    enable master ID replication
edma_channel_protection_level_t protectionLevel
    protection level
struct __edma_transfer_config
    #include <fsl_edma.h> edma4 channel transfer configuration
```

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

Note: User should pay attention to the transfer size alignment limitation

- a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
 - b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width
 - c. the totalBytes should align with the bytesEachRequest
 - d. the srcAddr should align with the srcWidthOfEachTransfer
 - e. the dstAddr should align with the dstWidthOfEachTransfer
 - f. the srcAddr should align with srcAddrModulo if modulo feature is enabled
 - g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.
-

Public Members

```
uint32_t srcAddr
    Source data address.
uint32_t destAddr
    Destination data address.
edma_transfer_size_t srcTransferSize
    Source data transfer size.
```

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

Sign-extended offset value in byte unit applied to the current source address to form the next-state value as each source read is completed

int16_t destOffset

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

uint32_t minorLoopBytes

bytes in each minor loop or each request range: 1 - (2^{30} - 1) when minor loop mapping is enabled range: 1 - (2^{10} - 1) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - (2^{32} - 1) when minor loop mapping is disabled

uint32_t majorLoopCounts

minor loop counts in each major loop, should be 1 at least for each transfer range: (0 - (2^{15} - 1)) when minor loop channel link is disabled range: (0 - (2^9 - 1)) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajorLoop * bytesEachMinorLoop

uint16_t enabledInterruptMask

channel interrupt to enable, can be OR'ed value of `_edma_interrupt_enable`

edma_modulo_t srcAddrModulo

source circular data queue range

int32_t srcMajorLoopOffset

source major loop offset

edma_modulo_t dstAddrModulo

destination circular data queue range

int32_t dstMajorLoopOffset

destination major loop offset

bool enableSrcMinorLoopOffset

enable source minor loop offset

bool enableDstMinorLoopOffset

enable dest minor loop offset

int32_t minorLoopOffset

burst offset, the offset will be applied after minor loop update

bool enableChannelMajorLoopLink

channel link when major loop complete

uint32_t majorLoopLinkChannel

major loop link channel number

bool enableChannelMinorLoopLink

channel link when minor loop complete

uint32_t minorLoopLinkChannel

minor loop link channel number

edma_tcd_t *linkTCD

pointer to the link transfer control descriptor

struct `_edma_config`

`#include <fsl_edma.h>` eDMA global configuration structure.

Public Members

bool 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.

2.5 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/dsiable 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

2.6 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)

2.7 EIM: error injection module

FSL_EIM_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.

2.8 EQDC: Enhanced Quadrature Encoder/Decoder

void EQDC_Init(EQDC_Type *base, const *eqdc_config_t* *psConfig)

Initializes the EQDC module.

This function initializes the EQDC by enabling the IP bus clock (optional).

Parameters

- base – EQDC peripheral base address.
- psConfig – Pointer to configuration structure.

void EQDC_GetDefaultConfig(*eqdc_config_t* *psConfig)

Gets an available pre-defined configuration.

The default value are:

```
psConfig->enableReverseDirection      = false;
psConfig->countOnce                    = false;
psConfig->operateMode                  = kEQDC_QuadratureDecodeOperationMode;
psConfig->countMode                    = kEQDC_QuadratureX4;
psConfig->homeEnableInitPosCounterMode = kEQDC_HomeInitPosCounterDisabled;
psConfig->indexPresetInitPosCounterMode = kEQDC_IndexInitPosCounterDisabled;
psConfig->enableIndexInitPositionCounter = false;
psConfig->enableDma                    = false;
psConfig->bufferedRegisterLoadMode     = false;
psConfig->enableTriggerInitPositionCounter = false;
psConfig->enableTriggerClearPositionRegisters = false;
psConfig->enableTriggerHoldPositionRegisters = false;
psConfig->enableWatchdog                = false;
psConfig->watchdogTimeoutValue          = 0xFFFFU;
psConfig->filterPhaseA                  = 0U;
psConfig->filterPhaseB                  = 0U;
psConfig->filterIndPre                   = 0U;
psConfig->filterHomEna                  = 0U;
psConfig->filterClockSourceSelection    = false;
psConfig->filterSampleCount              = kEQDC_Filter3Samples;
psConfig->filterSamplePeriod             = 0U;
psConfig->outputPulseMode                = kEQDC_OutputPulseOnCounterEqualCompare;
psConfig->positionCompareValue[0]        = 0xFFFFFFFFU;
psConfig->positionCompareValue[1]        = 0xFFFFFFFFU;
psConfig->positionCompareValue[2]        = 0xFFFFFFFFU;
psConfig->positionCompareValue[3]        = 0xFFFFFFFFU;
psConfig->revolutionCountCondition       = kEQDC_RevolutionCountOnIndexPulse;
psConfig->positionModulusValue           = 0U;
psConfig->positionInitialValue           = 0U;
psConfig->positionCounterValue           = 0U;
psConfig->enablePeriodMeasurement        = false;
psConfig->prescaler                      = kEQDC_Prescaler1;
psConfig->enabledInterruptsMask          = 0U;
```

Parameters

- psConfig – Pointer to configuration structure.

void EQDC_Deinit(EQDC_Type *base)

De-initializes the EQDC module.

This function deinitializes the EQDC by disabling the IP bus clock (optional).

Parameters

- base – EQDC peripheral base address.

void EQDC_SetOperateMode(EQDC_Type *base, *eqdc_operate_mode_t* operateMode)

Initializes the mode of operation.

This function initializes mode of operation by enabling the IP bus clock (optional).

Parameters

- `base` – EQDC peripheral base address.
- `operateMode` – Select operation mode.

```
static inline void EQDC_SetCountMode(EQDC_Type *base, eqdc_count_mode_t countMode)
```

Initializes the mode of count.

These bits control the basic counting and behavior of Position Counter and Position Difference Counter. Setting CTRL[REV] to 1 can reverse the counting direction. 1.In quadrature Mode (CTRL[PH1] = 0): 00b - CM0: Normal/Reverse Quadrature X4 01b - CM1: Normal/Reverse Quadrature X2 10b - CM2: Normal/Reverse Quadrature X1 11b - CM3: Reserved 2.In Single Phase Mode (CTRL[PH1] = 1): 00b - CM0: UP/DOWN Pulse Count Mode 01b - CM1: Signed Mode, count PHASEA rising/falling edge, position counter counts up when PHASEB is low and counts down when PHASEB is high 10b - CM2: Signed Count Mode, count PHASEA rising edge only, position counter counts up when PHASEB is low and counts down when PHASEB is high 11b - CM3: Reserved

Parameters

- `base` – EQDC peripheral base address.
- `countMode` – Select count mode.

```
static inline void EQDC_EnableWatchdog(EQDC_Type *base, bool bEnable)
```

Enable watchdog for EQDC module.

Parameters

- `base` – EQDC peripheral base address
- `bEnable` – Enables or disables the watchdog

```
static inline void EQDC_SetWatchdogTimeout(EQDC_Type *base, uint16_t u16Timeout)
```

Set watchdog timeout value.

Parameters

- `base` – EQDC peripheral base address
- `u16Timeout` – Number of clock cycles, plus one clock cycle that the watchdog timer counts before timing out

```
static inline void EQDC_EnableDMA(EQDC_Type *base, bool bEnable)
```

Enable DMA for EQDC module.

Parameters

- `base` – EQDC peripheral base address
- `bEnable` – Enables or disables the DMA

```
static inline void EQDC_SetBufferedRegisterLoadUpdateMode(EQDC_Type *base)
```

Set Buffered Register Load (Update) Mode.

This bit selects the loading time point of the buffered compare registers UCOMPx/LCOMPx, x=0~3, initial register (UNIT/LINIT), and modulus register (UMOD/LMOD). Buffered registers are loaded and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.

Parameters

- `base` – EQDC peripheral base address

```
static inline void EQDC_ClearBufferedRegisterLoadUpdateMode(EQDC_Type *base)
```

Clear Buffered Register Load (Update) Mode.

Buffered Register Load (Update) Mode bit selects the loading time point of the buffered compare registers UCOMPx/LCOMPx, x=0~3, initial register (UINIT/LINIT), and modulus register (UMOD/LMOD). Buffered registers are loaded and take effect immediately upon CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address

```
static inline void EQDC_SetEqdcLdok(EQDC_Type *base)
```

Set load okay.

Load okay enables that the outer-set values of buffered compare registers (UCOMPx/LCOMPx, x=0~3), initial register(UINIT/LINIT) and modulus register(UMOD/LMOD) can be loaded into their inner-sets and take effect. When LDOK is set, this loading action occurs at the next position counter roll-over or roll-under if CTRL2[LDMOD] is set, or it occurs immediately if CTRL2[LDMOD] is cleared. LDOK is automatically cleared after the values in outer-set is loaded into the inner-set.

Parameters

- base – EQDC peripheral base address.

```
static inline uint16_t EQDC_GetEqdcLdok(EQDC_Type *base)
```

Get load okay.

Parameters

- base – EQDC peripheral base address.

```
static inline void EQDC_ClearEqdcLdok(EQDC_Type *base)
```

Clear load okay.

Parameters

- base – EQDC peripheral base address.

```
static inline uint32_t EQDC_GetStatusFlags(EQDC_Type *base)
```

Get the status flags.

Parameters

- base – EQDC peripheral base address.

Returns

Logical OR'ed value of the status flags, `_eqdc_status_flags`.

```
static inline void EQDC_ClearStatusFlags(EQDC_Type *base, uint32_t u32Flags)
```

Clear the status flags.

Parameters

- base – EQDC peripheral base address.
- u32Flags – Logical OR'ed value of the flags to clear, `_eqdc_status_flags`.

```
static inline uint16_t EQDC_GetSignalStatusFlags(EQDC_Type *base)
```

Get the signals' real-time status.

Parameters

- base – EQDC peripheral base address.

Returns

Logical OR'ed value of the real-time signal status, `_eqdc_signal_status`.

static inline *eqdc_count_direction_flag_t* EQDC_GetLastCountDirection(EQDC_Type *base)
Get the direction of the last count.

Parameters

- base – EQDC peripheral base address.

Returns

Direction of the last count.

static inline void EQDC_EnableInterrupts(EQDC_Type *base, uint32_t u32Interrupts)
Enable the interrupts.

Parameters

- base – EQDC peripheral base address.
- u32Interrupts – Logical OR'ed value of the interrupts, *_eqdc_interrupt_enable*.

static inline void EQDC_DisableInterrupts(EQDC_Type *base, uint32_t u32Interrupts)
Disable the interrupts.

Parameters

- base – EQDC peripheral base address.
- u32Interrupts – Logical OR'ed value of the interrupts, *_eqdc_interrupt_enable*.

static inline void EQDC_DoSoftwareLoadInitialPositionValue(EQDC_Type *base)
Load the initial position value to position counter.

Software trigger to load 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 – EQDC peripheral base address.

static inline void EQDC_SetInitialPositionValue(EQDC_Type *base, uint32_t
u32PositionInitValue)

Set initial position value for EQDC module.

Set the position counter initial value (UNIT, LINIT). After writing values to the UNIT and LINIT registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionInitValue – Position initial value

static inline void EQDC_SetPositionCounterValue(EQDC_Type *base, uint32_t
positionCounterValue)

Set position counter value.

Set the position counter value (POS or UPOS, LPOS).

Parameters

- base – EQDC peripheral base address
- positionCounterValue – Position counter value

```
static inline void EQDC_SetPositionModulusValue(EQDC_Type *base, uint32_t
                                              positionModulusValue)
```

Set position counter modulus value.

Set the position counter modulus value (UMOD, LMOD). After writing values to the UMOD and LMOD registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- positionModulusValue – Position modulus value

```
static inline void EQDC_SetPositionCompare0Value(EQDC_Type *base, uint32_t
                                              u32PositionComp0Value)
```

Set position counter compare 0 value.

Set the position counter compare 0 value (UCOMP0, LCOMP0). After writing values to the UCOMP0 and LCOMP0 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp0Value – Position modulus value

```
static inline void EQDC_SetPositionCompare1Value(EQDC_Type *base, uint32_t
                                              u32PositionComp1Value)
```

Set position counter compare 1 value.

Set the position counter compare 1 value (UCOMP1, LCOMP1). After writing values to the UCOMP1 and LCOMP1 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp1Value – Position modulus value

```
static inline void EQDC_SetPositionCompare2Value(EQDC_Type *base, uint32_t
                                              u32PositionComp2Value)
```

Set position counter compare 2 value.

Set the position counter compare 2 value (UCOMP2, LCOMP2). After writing values to the UCOMP2 and LCOMP2 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp2Value – Position modulus value

```
static inline void EQDC_SetPositionCompare3Value(EQDC_Type *base, uint32_t
                                              u32PositionComp3Value)
```

Set position counter compare 3 value.

Set the position counter compare 3 value (UCOMP3, LCOMP3). After writing values to the UCOMP3 and LCOMP3 registers, the values are “buffered” into outer-set registers temporarily. Values will be loaded into inner-set registers and take effect using the following two methods:

- a. If CTRL2[LDMODE] is 1, “buffered” values are loaded into inner-set and take effect at the next roll-over or roll-under if CTRL[LDOK] is set.
- b. If CTRL2[LDMODE] is 0, “buffered” values are loaded into inner-set and take effect immediately when CTRL[LDOK] is set.

Parameters

- base – EQDC peripheral base address
- u32PositionComp3Value – Position modulus value

```
static inline uint32_t EQDC_GetPosition(EQDC_Type *base)
```

Get the current position counter’s value.

Parameters

- base – EQDC peripheral base address.

Returns

Current position counter’s value.

```
static inline uint32_t EQDC_GetHoldPosition(EQDC_Type *base)
```

Get the hold position counter’s value.

The position counter (POS or UPOS, LPOS) value is loaded to hold position (POSH or UPOSH, LPOSH) when:

- a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read.
- b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter's value.

```
static inline uint32_t EQDC_GetHoldPosition1(EQDC_Type *base)
```

Get the hold position counter1's value.

The Upper Position Counter Hold Register 1(UPOSH1) shares the same address with UCOMP1. When read, this register means the value of UPOSH1, which is the upper 16 bits of POSH1. The Lower Position Counter Hold Register 1(LPOSH1) shares the same address with LCOMP1. When read, this register means the value of LPOSH1, which is the lower 16 bits of POSH1. Position counter is captured into POSH1 on the rising edge of ICAP[1].

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter1's value.

```
static inline uint32_t EQDC_GetHoldPosition2(EQDC_Type *base)
```

Get the hold position counter2's value.

The Upper Position Counter Hold Register 2(UPOSH2) shares the same address with UCOMP2. When read, this register means the value of UPOSH2, which is the upper 16 bits of POSH2. The Lower Position Counter Hold Register 2(LPOSH2) shares the same address with LCOMP2. When read, this register means the value of LPOSH2, which is the lower 16 bits of POSH2. Position counter is captured into POSH2 on the rising edge of ICAP[2].

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter2's value.

```
static inline uint32_t EQDC_GetHoldPosition3(EQDC_Type *base)
```

Get the hold position counter3's value.

The Upper Position Counter Hold Register 3(UPOSH3) shares the same address with UCOMP3. When read, this register means the value of UPOSH3, which is the upper 16 bits of POSH3. The Lower Position Counter Hold Register 3(LPOSH3) shares the same address with LCOMP3. When read, this register means the value of LPOSH3, which is the lower 16 bits of POSH3. Position counter is captured into POSH3 on the rising edge of ICAP[3].

Parameters

- base – EQDC peripheral base address.

Returns

Hold position counter3's value.

```
static inline uint16_t EQDC_GetPositionDifference(EQDC_Type *base)
```

Get the position difference counter's value.

Parameters

- base – EQDC peripheral base address.

Returns

The position difference counter's value.

```
static inline uint16_t EQDC_GetHoldPositionDifference(EQDC_Type *base)
```

Get the hold position difference counter's value.

The position difference (POSD) value is loaded to hold position difference (POSDH) when:

- a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read. When Period Measurement is enabled (CTRL3[PMEN] = 1), POSDH will only be updated when reading POSD.
- b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

Parameters

- base – EQDC peripheral base address.

Returns

Hold position difference counter's value.

```
static inline uint16_t EQDC_GetRevolution(EQDC_Type *base)
```

Get the revolution counter's value.

Get the revolution counter (REV) value.

Parameters

- base – EQDC peripheral base address.

Returns

The revolution counter's value.

```
static inline uint16_t EQDC_GetHoldRevolution(EQDC_Type *base)
```

Get the hold revolution counter's value.

The revolution counter (REV) value is loaded to hold revolution (REXH) when:

- a. Position register (POS or UPOS, LPOS), or position difference register (POSD), or revolution register (REV) is read.
- b. TRIGGER happens and TRIGGER is enabled to update the hold registers.

Parameters

- base – EQDC peripheral base address.

Returns

Hold position revolution counter's value.

```
static inline uint16_t EQDC_GetLastEdgeTime(EQDC_Type *base)
```

Get the last edge time.

Last edge time (LASTEDGE) is the time since the last edge occurred on PHASEA or PHASEB. The last edge time register counts up using the peripheral clock after prescaler. Any edge on PHASEA or PHASEB will reset this register to 0 and start counting. If the last edge timer count reaches 0xffff, the counting will stop in order to prevent an overflow. Counting will continue when an edge occurs on PHASEA or PHASEB.

Parameters

- base – EQDC peripheral base address.

Returns

The last edge time.

```
static inline uint16_t EQDC_GetHoldLastEdgeTime(EQDC_Type *base)
```

Get the hold last edge time.

The hold of last edge time (LASTEDGEH) is update to last edge time (LASTEDGE) when the position difference register (POSD) is read.

Parameters

- base – EQDC peripheral base address.

Returns

Hold of last edge time.

static inline uint16_t EQDC_GetPositionDifferencePeriod(EQDC_Type *base)

Get the Position Difference Period counter value.

The Position Difference Period counter (POSDPER) counts up using the prescaled peripheral clock. When reading the position difference register(POSD), the last edge time (LAST-EDGE) will be loaded to position difference period counter(POSDPER). If the POSDPER count reaches 0xffff, the counting will stop in order to prevent an overflow. Counting will continue when an edge occurs on PHASEA or PHASEB.

Parameters

- base – EQDC peripheral base address.

Returns

The position difference period counter value.

static inline uint16_t EQDC_GetBufferedPositionDifferencePeriod(EQDC_Type *base)

Get buffered Position Difference Period counter value.

The Bufferd Position Difference Period (POSDPERBFR) value is updated with the position difference period counter(POSDPER) when any edge occurs on PHASEA or PHASEB.

Parameters

- base – EQDC peripheral base address.

Returns

The buffered position difference period counter value.

static inline uint16_t EQDC_GetHoldPositionDifferencePeriod(EQDC_Type *base)

Get Hold Position Difference Period counter value.

The hold position difference period(POSDPERH) is updated with the value of buffered position difference period(POSDPERBFR) when the position difference(POSD) register is read.

Parameters

- base – EQDC peripheral base address.

Returns

The hold position difference period counter value.

enum _eqdc_status_flags

EQDC status flags, these flags indicate the counter's events. .

Values:

enumerator kEQDC_HomeEnableTransitionFlag

HOME/ENABLE signal transition occurred.

enumerator kEQDC_IndexPresetPulseFlag

INDEX/PRESET pulse occurred.

enumerator kEQDC_WatchdogTimeoutFlag

Watchdog timeout occurred.

enumerator kEQDC_SimultPhaseChangeFlag

Simultaneous change of PHASEA and PHASEB occurred.

enumerator kEQDC_CountDirectionChangeFlag

Count direction change interrupt enable.

enumerator kEQDC_PositionRollOverFlag

Position counter rolls over from 0xFFFFFFFF to 0, or from MOD value to INIT value.

enumerator kEQDC_PositionRollUnderFlag

Position register roll under from 0 to 0xFFFFFFFF, or from INIT value to MOD value.

enumerator kEQDC_PositionCompare0Flag

Position counter match the COMP0 value.

enumerator kEQDC_PositionCompare1Flag

Position counter match the COMP1 value.

enumerator kEQDC_PositionCompare2Flag

Position counter match the COMP2 value.

enumerator kEQDC_PositionCompare3Flag

Position counter match the COMP3 value.

enumerator kEQDC_StatusAllFlags

enum _eqdc_signal_status

Signal status, these flags indicate the raw and filtered input signal status. .

Values:

enumerator kEQDC_SignalStatusRawHomeEnable

Raw HOME/ENABLE input.

enumerator kEQDC_SignalStatusRawIndexPreset

Raw INDEX/PRESET input.

enumerator kEQDC_SignalStatusRawPhaseB

Raw PHASEB input.

enumerator kEQDC_SignalStatusRawPhaseA

Raw PHASEA input.

enumerator kEQDC_SignalStatusFilteredHomeEnable

The filtered HOME/ENABLE input.

enumerator kEQDC_SignalStatusFilteredIndexPreset

The filtered INDEX/PRESET input.

enumerator kEQDC_SignalStatusFilteredPhaseB

The filtered PHASEB input.

enumerator kEQDC_SignalStatusFilteredPhaseA

The filtered PHASEA input.

enumerator kEQDC_SignalStatusPositionCompare0Flag

Position Compare 0 Flag Output.

enumerator kEQDC_SignalStatusPositionCompare1Flag

Position Compare 1 Flag Output.

enumerator kEQDC_SignalStatusPositionCompare2Flag

Position Compare 2 Flag Output.

enumerator kEQDC_SignalStatusPositionCompare3Flag

Position Compare 3 Flag Output.

enumerator kEQDC_SignalStatusCountDirectionFlagHold

Count Direction Flag Hold.

enumerator kEQDC_SignalStatusCountDirectionFlag

Count Direction Flag Output.

enumerator kEQDC_SignalStatusAllFlags

enum _eqdc_interrupt_enable

Interrupt enable/disable mask. .

Values:

enumerator kEQDC_HomeEnableTransitionInterruptEnable

HOME/ENABLE signal transition interrupt enable.

enumerator kEQDC_IndexPresetPulseInterruptEnable

INDEX/PRESET pulse interrupt enable.

enumerator kEQDC_WatchdogTimeoutInterruptEnable

Watchdog timeout interrupt enable.

enumerator kEQDC_SimultPhaseChangeInterruptEnable

Simultaneous PHASEA and PHASEB change interrupt enable.

enumerator kEQDC_CountDirectionChangeInterruptEnable

Count direction change interrupt enable.

enumerator kEQDC_PositionRollOverInterruptEnable

Roll-over interrupt enable.

enumerator kEQDC_PositionRollUnderInterruptEnable

Roll-under interrupt enable.

enumerator kEQDC_PositionCompare0InterruptEnable

Position compare 0 interrupt enable.

enumerator kEQDC_PositionCompare1InterruptEnable

Position compare 1 interrupt enable.

enumerator kEQDC_PositionCompare2InterruptEnable

Position compare 2 interrupt enable.

enumerator kEQDC_PositionCompare3InterruptEnable

Position compare 3 interrupt enable.

enumerator kEQDC_AllInterruptEnable

enum _eqdc_home_enable_init_pos_counter_mode

Define HOME/ENABLE signal's trigger mode.

Values:

enumerator kEQDC_HomeInitPosCounterDisabled

Don't use HOME/ENABLE signal to initialize the position counter.

enumerator kEQDC_HomeInitPosCounterOnRisingEdge

Use positive going edge to trigger initialization of position counters.

enumerator kEQDC_HomeInitPosCounterOnFallingEdge

Use negative going edge to trigger initialization of position counters.

enum _eqdc_index_preset_init_pos_counter_mode

Define INDEX/PRESET signal's trigger mode.

Values:

enumerator kEQDC_IndexInitPosCounterDisabled

INDEX/PRESET pulse does not initialize the position counter.

enumerator kEQDC_IndexInitPosCounterOnRisingEdge

Use INDEX/PRESET pulse rising edge to initialize position counter.

enumerator kEQDC_IndexInitPosCounterOnFallingEdge

Use INDEX/PRESET pulse falling edge to initialize position counter.

enum _eqdc_operate_mode

Define type for decoder operation mode.

The Quadrature Decoder operates in following 4 operation modes: 1.Quadrature Decode(QDC) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 0) In QDC operation mode, Module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 2.Quadrature Count(QCT) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 1) In QCT operation mode, Module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor. 3.Single Phase Decode(PH1DC) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 0) In PH1DC operation mode, the module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 4.Single Phase Count(PH1CT) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 1) In PH1CT operation mode, the module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor.

Values:

enumerator kEQDC_QuadratureDecodeOperationMode

Use standard quadrature decoder with PHASEA/PHASEB, INDEX/HOME.

enumerator kEQDC_QuadratureCountOperationMode

Use quadrature count operation mode with PHASEA/PHASEB, PRESET/ENABLE.

enumerator kEQDC_SinglePhaseDecodeOperationMode

Use single phase quadrature decoder with PHASEA/PHASEB, INDEX/HOME.

enumerator kEQDC_SinglePhaseCountOperationMode

Use single phase count decoder with PHASEA/PHASEB, PRESET/ENABLE.

enum _eqdc_count_mode

Define type for decoder count mode.

In decode mode, it uses the standard quadrature decoder with PHASEA and PHASEB, PHASEA = 0 and PHASEB = 0 mean reverse direction.

- If PHASEA leads PHASEB, then motion is in the positive direction.
- If PHASEA trails PHASEB, then motion is in the negative direction. In single phase mode, there are three count modes:
- In Signed Count mode (Single Edge). Both position counter (POS) and position difference counter (POSD) count on the input PHASEA rising edge while the input PHASEB provides the selected position counter direction (up/down). If CTRL[REV] is 1, then the position counter will count in the opposite direction.
- In Signed Count mode (double edge), both position counter (POS) and position difference counter (POSD) count the input PHASEA on both rising edge and falling edge while the input PHASEB provides the selected position counter direction (up/down).
- In UP/DOWN Pulse Count mode. Both position counter (POS) and position difference counter (POSD) count in the up direction when input PHASEA rising edge occurs. Both counters count in the down direction when input PHASEB rising edge occurs. If CTRL[REV] is 1, then the position counter will count in the opposite direction.

Values:

enumerator kEQDC_QuadratureX4

Active on kEQDC_QuadratureDecodeOperationMode/kEQDC_QuadratureCountOperationMode.

enumerator kEQDC_QuadratureX2

Active on kEQDC_QuadratureDecodeOperationMode/kEQDC_QuadratureCountOperationMode.

enumerator kEQDC_QuadratureX1

Active on kEQDC_QuadratureDecodeOperationMode/kEQDC_QuadratureCountOperationMode.

enumerator kEQDC_UpDownPulseCount

Active on kEQDC_SinglePhaseDecodeOperationMode/kEQDC_SinglePhaseCountOperationMode.

enumerator kEQDC_SignedCountDoubleEdge

Active on kEQDC_SinglePhaseDecodeOperationMode/kEQDC_SinglePhaseCountOperationMode.

enumerator kEQDC_SignedCountSingleEdge

Active on kEQDC_SinglePhaseDecodeOperationMode/kEQDC_SinglePhaseCountOperationMode.

enum _eqdc_output_pulse_mode

Define type for the condition of POSMATCH pulses.

Values:

enumerator kEQDC_OutputPulseOnCounterEqualCompare

POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (UCOMPx/LCOMPx)(x range is 0-3).

enumerator kEQDC_OutputPulseOnReadingPositionCounter

POSMATCH pulses when reading position counter(POS and LPOS), revolution counter(REV), position difference counter(POSD).

enum _eqdc_revolution_count_condition

Define type for determining how the revolution counter (REV) is incremented/decremented.

Values:

enumerator kEQDC_RevolutionCountOnIndexPulse

Use INDEX pulse to increment/decrement revolution counter.

enumerator kEQDC_RevolutionCountOnRollOverModulus

Use modulus counting roll-over/under to increment/decrement revolution counter.

enum _eqdc_filter_sample_count

Input Filter Sample Count.

The Input Filter Sample Count represents the number of consecutive samples that must agree, before the input filter accepts an input transition

Values:

enumerator kEQDC_Filter3Samples

3 samples.

enumerator kEQDC_Filter4Samples

4 samples.

enumerator kEQDC_Filter5Samples

5 samples.

enumerator kEQDC_Filter6Samples

6 samples.

enumerator kEQDC_Filter7Samples
7 samples.

enumerator kEQDC_Filter8Samples
8 samples.

enumerator kEQDC_Filter9Samples
9 samples.

enumerator kEQDC_Filter10Samples
10 samples.

enum _eqdc_count_direction_flag
Count direction.

Values:

enumerator kEQDC_CountDirectionDown
Last count was in down direction.

enumerator kEQDC_CountDirectionUp
Last count was in up direction.

enum _eqdc_prescaler
Prescaler used by Last Edge Time (LASTEDGE) and Position Difference Period Counter (POS-DPER).

Values:

enumerator kEQDC_Prescaler1
Prescaler value 1.

enumerator kEQDC_Prescaler2
Prescaler value 2.

enumerator kEQDC_Prescaler4
Prescaler value 4.

enumerator kEQDC_Prescaler8
Prescaler value 8.

enumerator kEQDC_Prescaler16
Prescaler value 16.

enumerator kEQDC_Prescaler32
Prescaler value 32.

enumerator kEQDC_Prescaler64
Prescaler value 64.

enumerator kEQDC_Prescaler128
Prescaler value 128.

enumerator kEQDC_Prescaler256
Prescaler value 256.

enumerator kEQDC_Prescaler512
Prescaler value 512.

enumerator kEQDC_Prescaler1024
Prescaler value 1024.

enumerator kEQDC_Prescaler2048
Prescaler value 2048.

enumerator kEQDC_Prescaler4096

Prescaler value 4096.

enumerator kEQDC_Prescaler8192

Prescaler value 8192.

enumerator kEQDC_Prescaler16384

Prescaler value 16384.

enumerator kEQDC_Prescaler32768

Prescaler value 32768.

typedef enum *_eqdc_home_enable_init_pos_counter_mode*

eqdc_home_enable_init_pos_counter_mode_t

Define HOME/ENABLE signal's trigger mode.

typedef enum *_eqdc_index_preset_init_pos_counter_mode*

eqdc_index_preset_init_pos_counter_mode_t

Define INDEX/PRESET signal's trigger mode.

typedef enum *_eqdc_operate_mode* *eqdc_operate_mode_t*

Define type for decoder operation mode.

The Quadrature Decoder operates in following 4 operation modes: 1.Quadrature Decode(QDC) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 0) In QDC operation mode, Module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 2.Quadrature Count(QCT) Operation Mode (CTRL[PH1] = 0,CTRL2[OPMODE] = 1) In QCT operation mode, Module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor. 3.Single Phase Decode(PH1DC) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 0) In PH1DC operation mode, the module uses PHASEA, PHASEB, INDEX, HOME, TRIGGER and ICAP[3:1] to decode the PHASEA and PHASEB signals from Speed/Position sensor. 4.Single Phase Count(PH1CT) Operation Mode (CTRL[PH1] = 1,CTRL2[OPMODE] = 1) In PH1CT operation mode, the module uses PHASEA, PHASEB, PRESET, ENABLE, TRIGGER and ICAP[3:1] to count the PHASEA and PHASEB signals from Speed/Position sensor.

typedef enum *_eqdc_count_mode* *eqdc_count_mode_t*

Define type for decoder count mode.

In decode mode, it uses the standard quadrature decoder with PHASEA and PHASEB, PHASEA = 0 and PHASEB = 0 mean reverse direction.

- If PHASEA leads PHASEB, then motion is in the positive direction.
- If PHASEA trails PHASEB, then motion is in the negative direction. In single phase mode, there are three count modes:
- In Signed Count mode (Single Edge). Both position counter (POS) and position difference counter (POSD) count on the input PHASEA rising edge while the input PHASEB provides the selected position counter direction (up/down). If CTRL[REV] is 1, then the position counter will count in the opposite direction.
- In Signed Count mode (double edge), both position counter (POS) and position difference counter (POSD) count the input PHASEA on both rising edge and falling edge while the input PHASEB provides the selected position counter direction (up/down).
- In UP/DOWN Pulse Count mode. Both position counter (POS) and position difference counter (POSD) count in the up direction when input PHASEA rising edge occurs. Both counters count in the down direction when input PHASEB rising edge occurs. If CTRL[REV] is 1, then the position counter will count in the opposite direction.

```
typedef enum _eqdc_output_pulse_mode eqdc_output_pulse_mode_t
```

Define type for the condition of POSMATCH pulses.

```
typedef enum _eqdc_revolution_count_condition eqdc_revolution_count_condition_t
```

Define type for determining how the revolution counter (REV) is incremented/decremented.

```
typedef enum _eqdc_filter_sample_count eqdc_filter_sample_count_t
```

Input Filter Sample Count.

The Input Filter Sample Count represents the number of consecutive samples that must agree, before the input filter accepts an input transition

```
typedef enum _eqdc_count_direction_flag eqdc_count_direction_flag_t
```

Count direction.

```
typedef enum _eqdc_prescaler eqdc_prescaler_t
```

Prescaler used by Last Edge Time (LASTEDGE) and Position Difference Period Counter (POS-DPER).

```
typedef struct _eqdc_config eqdc_config_t
```

Define user configuration structure for EQDC module.

```
FSL_EQDC_DRIVER_VERSION
```

```
EQDC_CTRL_W1C_FLAGS
```

W1C bits in EQDC CTRL registers.

```
EQDC_INTCTRL_W1C_FLAGS
```

W1C bits in EQDC INTCTRL registers.

```
EQDC_CTRL_INT_EN
```

Interrupt enable bits in EQDC CTRL registers.

```
EQDC_INTCTRL_INT_EN
```

Interrupt enable bits in EQDC INTCTRL registers.

```
EQDC_CTRL_INT_FLAGS
```

Interrupt flag bits in EQDC CTRL registers.

```
EQDC_INTCTRL_INT_FLAGS
```

Interrupt flag bits in EQDC INTCTRL registers.

```
kEQDC_PositionCompare0InterruptEnable
```

```
kEQDC_PositionCompare1InterruptEnable
```

```
kEQDC_PositionCompare2InterruptEnable
```

```
kEQDC_PositionCompare3InterruptEnable
```

```
struct _eqdc_config
```

#include <fsl_eqdc.h> Define user configuration structure for EQDC module.

Public Members

```
bool enableReverseDirection
```

Enable reverse direction counting.

```
bool countOnce
```

Selects modulo loop or one shot counting mode.

`bool enableDma`

Enable DMA for new written buffer values of COMPx/INIT/MOD(x range is 0-3)

`bool bufferedRegisterLoadMode`

selects the loading time point of the buffered compare registers UCOMPx/LCOMPx, x=0~3, initial register (UINIT/LINIT), and modulus register (UMOD/LMOD).

`bool enableTriggerInitPositionCounter`

Initialize position counter with initial register(UINIT, LINIT) value on TRIGGER's rising edge.

`bool enableTriggerClearPositionRegisters`

Clear position counter(POS), revolution counter(REV), position difference counter (POSD) on TRIGGER's rising edge.

`bool enableTriggerHoldPositionRegisters`

Load position counter(POS), revolution counter(REV), position difference counter (POSD) values to hold registers on TRIGGER's rising edge.

`bool filterPhaseA`

Filter operation on PHASEA input, when write 1, it means filter for PHASEA input is bypassed.

`bool filterPhaseB`

Filter operation on PHASEB input, when write 1, it means filter for PHASEB input is bypassed.

`bool filterIndPre`

Filter operation on INDEX/PRESET input, when write 1, it means filter for INDEX/PRESET input is bypassed.

`bool filterHomEna`

Filter operation on HOME/ENABLE input, when write 1, it means filter for HOME/ENABLE input is bypassed.

`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.

`eqdc_prescaler_t prescaler`

Prescaler.

`bool filterClockSourceselection`

Filter Clock Source selection.

`eqdc_filter_sample_count_t filterSampleCount`

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.

`uint8_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.

`eqdc_operate_mode_t operateMode`

Selects operation mode.

eqdc_count_mode_t countMode

Selects count mode.

eqdc_home_enable_init_pos_counter_mode_t homeEnableInitPosCounterMode

Select how HOME/Enable signal used to initialize position counters.

eqdc_index_preset_init_pos_counter_mode_t indexPresetInitPosCounterMode

Select how INDEX/Preset signal used to initialize position counters.

eqdc_output_pulse_mode_t outputPulseMode

The condition of POSMATCH pulses.

uint32_t positionCompareValue[4]

Position compare 0 ~ 3 value. The available value is a 32-bit number.

eqdc_revolution_count_condition_t revolutionCountCondition

Revolution Counter Modulus Enable.

uint32_t positionModulusValue

Position modulus value. The available value is a 32-bit number.

uint32_t positionInitialValue

Position initial value. The available value is a 32-bit number.

uint32_t positionCounterValue

Position counter value. When Modulo mode enabled, the positionCounterValue should be in the range of positionInitialValue and positionModulusValue.

bool enablePeriodMeasurement

Enable period measurement. When enabled, the position difference hold register (POSDH) is only updated when position difference register (POSD) is read.

uint16_t enabledInterruptsMask

Mask of interrupts to be enabled, should be OR'ed value of *_eqdc_interrupt_enable*.

2.9 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.

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.

2.10 FGPIO Driver

2.11 C40ESP3 Flash Driver

enum _flash_driver_api_keys
Enumeration for Flash driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFLASH_ApiEraseKey
Key value used to validate all flash erase APIs.

status_t FLASH_Init(*flash_config_t* *config)
Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.

status_t FLASH_Erase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

status_t FLASH_EraseAll(FMU_Type *base, uint32_t key)
Erases entire flash and ifr.

status_t FLASH_Program(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash phrases with data at locations passed in through parameters.

status_t FLASH_ProgramPage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash pages with data at locations passed in through parameters.

status_t FLASH_VerifyErasePhrase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash phrases are erased.

status_t FLASH_VerifyErasePage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash pages are erased.

status_t FLASH_VerifyEraseSector(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash sectors are erased.

status_t FLASH_VerifyEraseAll(FMU_Type *base)

Verify that all flash and IFR space is erased.

status_t FLASH_VerifyEraseBlock(*flash_config_t* *config, FMU_Type *base, uint32_t blockaddr)

Verify that a flash block is erased.

status_t FLASH_VerifyEraseIFRPhrase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr phrases are erased.

status_t FLASH_VerifyEraseIFRPage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr pages are erased.

status_t FLASH_VerifyEraseIFRSector(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr sectors are erased.

status_t FLASH_GetProperty(*flash_config_t* *config, *flash_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flash property.

status_t Read_Into_MISR(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t ending, uint32_t *seed, uint32_t *signature)

Read into MISR.

The Read into MISR operation generates a signature based on the contents of the selected flash memory using an embedded MISR.

status_t Read_IFR_Into_MISR(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t ending, uint32_t *seed, uint32_t *signature)

Read IFR into MISR.

The Read IFR into MISR operation generates a signature based on the contents of the selected IFR space using an embedded MISR.

FSL_FLASH_DRIVER_VERSION

Flash driver version for SDK.

Version 2.3.3.

enum _flash_driver_version_constants

Flash driver version for ROM.

Values:

enumerator kFLASH_DriverVersionName

Flash driver version name.

enumerator kFLASH_DriverVersionMajor

Major flash driver version.

enumerator kFLASH_DriverVersionMinor
Minor flash driver version.

enumerator kFLASH_DriverVersionBugfix
Bugfix for flash driver version.

enum _flash_property_tag
Enumeration for various flash properties.

Values:

enumerator kFLASH_PropertyPflash0SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash0TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash0BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash0BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash0BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash0FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash0AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash0AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyPflash1SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash1TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash1BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash1BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash1BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash1FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash1AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash1AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyFlexRamBlockBaseAddr
FlexRam block base address property.

enumerator kFLASH_PropertyFlexRamTotalSize

FlexRam total size property.

typedef enum *_flash_property_tag* flash__property_tag__t

Enumeration for various flash properties.

typedef struct *_flash_mem_descriptor* flash__mem_desc__t

Flash memory descriptor.

typedef struct *_flash_ifr_desc* flash__ifr_desc__t

typedef struct *_msf1_config* msf1__config__t

typedef struct *_flash_config* flash__config__t

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

FLASH__ADDR__MASK

struct *_flash_mem_descriptor*

#include <fsl_k4_flash.h> Flash memory descriptor.

Public Members

uint32_t blockBase

Base address of the flash block

uint32_t totalSize

The size of the flash block.

uint32_t blockCount

A number of flash blocks.

struct *_flash_ifr_desc*

#include <fsl_k4_flash.h>

struct *_msf1_config*

#include <fsl_k4_flash.h>

struct *_flash_config*

#include <fsl_k4_flash.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

2.12 FREQME: Frequency Measurement

2.13 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.

2.14 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)
```

Denitalizes 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 *verInfo)`

Get GPIO version information.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- verInfo – 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

GPIO__PortGetInterruptFlags(base)

GPIO__PortClearInterruptFlags(base, mask)

2.15 INPUTMUX: Input Multiplexing Driver

enum _inputmux_index_t

Values:

enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL0

enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL1

enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL2

enumerator kINPUTMUX_INDEX_CTIMER0CAPTSEL3

enumerator kINPUTMUX_INDEX_CTIMER1CAPTSEL0

enumerator kINPUTMUX_INDEX_CTIMER1CAPTSEL1

enumerator kINPUTMUX_INDEX_CTIMER1CAPTSEL2

enumerator kINPUTMUX_INDEX_CTIMER1CAPTSEL3

enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL0

enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL1

enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL2

enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL3
enumerator kINPUTMUX_INDEX_FLEXPWM0_FAULTSEL0
enumerator kINPUTMUX_INDEX_FLEXPWM0_FAULTSEL1
enumerator kINPUTMUX_INDEX_FLEXPWM0_FAULTSEL2
enumerator kINPUTMUX_INDEX_FLEXPWM0_FAULTSEL3
enumerator kINPUTMUX_INDEX_FLEXPWM1_FAULTSEL0
enumerator kINPUTMUX_INDEX_FLEXPWM1_FAULTSEL1
enumerator kINPUTMUX_INDEX_FLEXPWM1_FAULTSEL2
enumerator kINPUTMUX_INDEX_FLEXPWM1_FAULTSEL3
enumerator kINPUTMUX_INDEX_EXT_TRIGSEL0
enumerator kINPUTMUX_INDEX_EXT_TRIGSEL1
enumerator kINPUTMUX_INDEX_EXT_TRIGSEL2
enumerator kINPUTMUX_INDEX_EXT_TRIGSEL3
enumerator kINPUTMUX_INDEX_EXT_TRIGSEL4
enumerator kINPUTMUX_INDEX_EXT_TRIGSEL6
enumerator kINPUTMUX_INDEX_EXT_TRIGSEL7

enum __inputmux_connection_t
INPUTMUX connections type.

Values:

enumerator kINPUTMUX_Ctinp0ToCtimer0Cap
enumerator kINPUTMUX_Ctinp1ToCtimer0Cap
enumerator kINPUTMUX_Ctinp2ToCtimer0Cap
enumerator kINPUTMUX_Ctinp3ToCtimer0Cap
enumerator kINPUTMUX_Ctinp4ToCtimer0Cap
enumerator kINPUTMUX_Ctinp5ToCtimer0Cap
enumerator kINPUTMUX_Ctinp6ToCtimer0Cap
enumerator kINPUTMUX_Ctinp8ToCtimer0Cap
enumerator kINPUTMUX_Ctinp9ToCtimer0Cap
enumerator kINPUTMUX_Ctinp12ToCtimer0Cap
enumerator kINPUTMUX_Ctinp13ToCtimer0Cap
enumerator kINPUTMUX_Ctinp16ToCtimer0Cap
enumerator kINPUTMUX_Ctinp17ToCtimer0Cap
enumerator kINPUTMUX_Ctinp18ToCtimer0Cap
enumerator kINPUTMUX_Ctinp19ToCtimer0Cap

enumerator kINPUTMUX__Adc0Tcomp0ToCtimer0Cap
enumerator kINPUTMUX__Adc0Tcomp1ToCtimer0Cap
enumerator kINPUTMUX__Adc0Tcomp2ToCtimer0Cap
enumerator kINPUTMUX__Adc0Tcomp3ToCtimer0Cap
enumerator kINPUTMUX__Cmp0OutToCtimer0Cap
enumerator kINPUTMUX__Ctimer1Mat1ToCtimer0Cap
enumerator kINPUTMUX__Ctimer1Mat2ToCtimer0Cap
enumerator kINPUTMUX__Ctimer1Mat3ToCtimer0Cap
enumerator kINPUTMUX__Pwm0Sm0Muxtrig0ToCtimer0Cap
enumerator kINPUTMUX__Pwm0Sm1Muxtrig0ToCtimer0Cap
enumerator kINPUTMUX__Pwm0Sm2Muxtrig0ToCtimer0Cap
enumerator kINPUTMUX__Lpi2C0MasterendofpacketToCtimer0Cap
enumerator kINPUTMUX__Lpi2C0SlaveendofpacketToCtimer0Cap
enumerator kINPUTMUX__Lpspi0EndofframeToCtimer0Cap
enumerator kINPUTMUX__Lpspi0ReceiveddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart0ReceiveddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart0TransmitteddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart0ReceivelineidleToCtimer0Cap
enumerator kINPUTMUX__Lpuart1ReceiveddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart1TransmitteddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart1ReceivelineidleToCtimer0Cap
enumerator kINPUTMUX__Lpuart2ReceiveddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart2TransmitteddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart2ReceivelineidleToCtimer0Cap
enumerator kINPUTMUX__Trigin0ToCtimer0Cap
enumerator kINPUTMUX__Trigin1ToCtimer0Cap
enumerator kINPUTMUX__Trigin2ToCtimer0Cap
enumerator kINPUTMUX__Trigin3ToCtimer0Cap
enumerator kINPUTMUX__Trigin4ToCtimer0Cap
enumerator kINPUTMUX__Trigin5ToCtimer0Cap
enumerator kINPUTMUX__Trigin6ToCtimer0Cap
enumerator kINPUTMUX__Trigin7ToCtimer0Cap
enumerator kINPUTMUX__Trigin8ToCtimer0Cap

enumerator kINPUTMUX__Trigin9ToCtimer0Cap
enumerator kINPUTMUX__Trigin10ToCtimer0Cap
enumerator kINPUTMUX__Trigin11ToCtimer0Cap
enumerator kINPUTMUX__Lpuart3ReceiveddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart3TransmitteddatawordToCtimer0Cap
enumerator kINPUTMUX__Lpuart3ReceivelineidleToCtimer0Cap
enumerator kINPUTMUX__Ctinp0ToCtimer1Cap
enumerator kINPUTMUX__Ctinp1ToCtimer1Cap
enumerator kINPUTMUX__Ctinp2ToCtimer1Cap
enumerator kINPUTMUX__Ctinp3ToCtimer1Cap
enumerator kINPUTMUX__Ctinp4ToCtimer1Cap
enumerator kINPUTMUX__Ctinp5ToCtimer1Cap
enumerator kINPUTMUX__Ctinp6ToCtimer1Cap
enumerator kINPUTMUX__Ctinp8ToCtimer1Cap
enumerator kINPUTMUX__Ctinp9ToCtimer1Cap
enumerator kINPUTMUX__Ctinp12ToCtimer1Cap
enumerator kINPUTMUX__Ctinp13ToCtimer1Cap
enumerator kINPUTMUX__Ctinp16ToCtimer1Cap
enumerator kINPUTMUX__Ctinp17ToCtimer1Cap
enumerator kINPUTMUX__Ctinp18ToCtimer1Cap
enumerator kINPUTMUX__Ctinp19ToCtimer1Cap
enumerator kINPUTMUX__Adc0Tcomp0ToCtimer1Cap
enumerator kINPUTMUX__Adc0Tcomp1ToCtimer1Cap
enumerator kINPUTMUX__Adc0Tcomp2ToCtimer1Cap
enumerator kINPUTMUX__Adc0Tcomp3ToCtimer1Cap
enumerator kINPUTMUX__Cmp0OutToCtimer1Cap
enumerator kINPUTMUX__Ctimer0Mat1ToCtimer1Cap
enumerator kINPUTMUX__Ctimer0Mat2ToCtimer1Cap
enumerator kINPUTMUX__Ctimer0Mat3ToCtimer1Cap
enumerator kINPUTMUX__Pwm0Sm0Muxtrig0ToCtimer1Cap
enumerator kINPUTMUX__Pwm0Sm1Muxtrig0ToCtimer1Cap
enumerator kINPUTMUX__Pwm0Sm2Muxtrig0ToCtimer1Cap
enumerator kINPUTMUX__Lpi2C0MasterendofpacketToCtimer1Cap

enumerator kINPUTMUX_Lpi2C0SlaveendofpacketToCtimer1Cap
enumerator kINPUTMUX_Lpspi0EndofframeToCtimer1Cap
enumerator kINPUTMUX_Lpspi0ReceiveddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart0ReceiveddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart0TransmitteddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart0ReceivelineidleToCtimer1Cap
enumerator kINPUTMUX_Lpuart1ReceiveddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart1TransmitteddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart1ReceivelineidleToCtimer1Cap
enumerator kINPUTMUX_Lpuart2ReceiveddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart2TransmitteddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart2ReceivelineidleToCtimer1Cap
enumerator kINPUTMUX_Trigin0ToCtimer1Cap
enumerator kINPUTMUX_Trigin1ToCtimer1Cap
enumerator kINPUTMUX_Trigin2ToCtimer1Cap
enumerator kINPUTMUX_Trigin3ToCtimer1Cap
enumerator kINPUTMUX_Trigin4ToCtimer1Cap
enumerator kINPUTMUX_Trigin5ToCtimer1Cap
enumerator kINPUTMUX_Trigin6ToCtimer1Cap
enumerator kINPUTMUX_Trigin7ToCtimer1Cap
enumerator kINPUTMUX_Trigin8ToCtimer1Cap
enumerator kINPUTMUX_Trigin9ToCtimer1Cap
enumerator kINPUTMUX_Trigin10ToCtimer1Cap
enumerator kINPUTMUX_Trigin11ToCtimer1Cap
enumerator kINPUTMUX_Lpuart3ReceiveddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart3TransmitteddatawordToCtimer1Cap
enumerator kINPUTMUX_Lpuart3ReceivelineidleToCtimer1Cap
enumerator kINPUTMUX_Ctinp0ToCtimer0Trig
enumerator kINPUTMUX_Ctinp1ToCtimer0Trig
enumerator kINPUTMUX_Ctinp2ToCtimer0Trig
enumerator kINPUTMUX_Ctinp3ToCtimer0Trig
enumerator kINPUTMUX_Ctinp4ToCtimer0Trig
enumerator kINPUTMUX_Ctinp5ToCtimer0Trig

enumerator kINPUTMUX__Ctinp6ToCtimer0Trig
enumerator kINPUTMUX__Ctinp8ToCtimer0Trig
enumerator kINPUTMUX__Ctinp9ToCtimer0Trig
enumerator kINPUTMUX__Ctinp12ToCtimer0Trig
enumerator kINPUTMUX__Ctinp13ToCtimer0Trig
enumerator kINPUTMUX__Ctinp16ToCtimer0Trig
enumerator kINPUTMUX__Ctinp17ToCtimer0Trig
enumerator kINPUTMUX__Ctinp18ToCtimer0Trig
enumerator kINPUTMUX__Ctinp19ToCtimer0Trig
enumerator kINPUTMUX__Adc0Tcomp0ToCtimer0Trig
enumerator kINPUTMUX__Adc0Tcomp1ToCtimer0Trig
enumerator kINPUTMUX__Adc0Tcomp2ToCtimer0Trig
enumerator kINPUTMUX__Adc0Tcomp3ToCtimer0Trig
enumerator kINPUTMUX__Cmp0OutToCtimer0Trig
enumerator kINPUTMUX__Ctimer1Mat1ToCtimer0Trig
enumerator kINPUTMUX__Ctimer1Mat2ToCtimer0Trig
enumerator kINPUTMUX__Ctimer1Mat3ToCtimer0Trig
enumerator kINPUTMUX__Pwm0Sm0Muxtrig0ToCtimer0Trig
enumerator kINPUTMUX__Pwm0Sm1Muxtrig0ToCtimer0Trig
enumerator kINPUTMUX__Pwm0Sm2Muxtrig0ToCtimer0Trig
enumerator kINPUTMUX__Lpi2C0MasterendofpacketToCtimer0Trig
enumerator kINPUTMUX__Lpi2C0SlaveendofpacketToCtimer0Trig
enumerator kINPUTMUX__Lpspi0EndofframeToCtimer0Trig
enumerator kINPUTMUX__Lpspi0ReceiveddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart0ReceiveddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart0TransmitteddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart0ReceivelineidleToCtimer0Trig
enumerator kINPUTMUX__Lpuart1ReceiveddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart1TransmitteddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart1ReceivelineidleToCtimer0Trig
enumerator kINPUTMUX__Lpuart2ReceiveddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart2TransmitteddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart2ReceivelineidleToCtimer0Trig

enumerator kINPUTMUX__Trigin0ToCtimer0Trig
enumerator kINPUTMUX__Trigin1ToCtimer0Trig
enumerator kINPUTMUX__Trigin2ToCtimer0Trig
enumerator kINPUTMUX__Trigin3ToCtimer0Trig
enumerator kINPUTMUX__Trigin4ToCtimer0Trig
enumerator kINPUTMUX__Trigin5ToCtimer0Trig
enumerator kINPUTMUX__Trigin6ToCtimer0Trig
enumerator kINPUTMUX__Trigin7ToCtimer0Trig
enumerator kINPUTMUX__Trigin8ToCtimer0Trig
enumerator kINPUTMUX__Trigin9ToCtimer0Trig
enumerator kINPUTMUX__Trigin10ToCtimer0Trig
enumerator kINPUTMUX__Trigin11ToCtimer0Trig
enumerator kINPUTMUX__Lpuart3ReceiveddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart3TransmitteddatawordToCtimer0Trig
enumerator kINPUTMUX__Lpuart3ReceivelineidleToCtimer0Trig
enumerator kINPUTMUX__Ctinp0ToCtimer1Trig
enumerator kINPUTMUX__Ctinp1ToCtimer1Trig
enumerator kINPUTMUX__Ctinp2ToCtimer1Trig
enumerator kINPUTMUX__Ctinp3ToCtimer1Trig
enumerator kINPUTMUX__Ctinp4ToCtimer1Trig
enumerator kINPUTMUX__Ctinp5ToCtimer1Trig
enumerator kINPUTMUX__Ctinp6ToCtimer1Trig
enumerator kINPUTMUX__Ctinp8ToCtimer1Trig
enumerator kINPUTMUX__Ctinp9ToCtimer1Trig
enumerator kINPUTMUX__Ctinp12ToCtimer1Trig
enumerator kINPUTMUX__Ctinp13ToCtimer1Trig
enumerator kINPUTMUX__Ctinp16ToCtimer1Trig
enumerator kINPUTMUX__Ctinp17ToCtimer1Trig
enumerator kINPUTMUX__Ctinp18ToCtimer1Trig
enumerator kINPUTMUX__Ctinp19ToCtimer1Trig
enumerator kINPUTMUX__Adc0Tcomp0ToCtimer1Trig
enumerator kINPUTMUX__Adc0Tcomp1ToCtimer1Trig
enumerator kINPUTMUX__Adc0Tcomp2ToCtimer1Trig

enumerator kINPUTMUX__Adc0Tcomp3ToCtimer1Trig
enumerator kINPUTMUX__Cmp0OutToCtimer1Trig
enumerator kINPUTMUX__Ctimer0Mat1ToCtimer1Trig
enumerator kINPUTMUX__Ctimer0Mat2ToCtimer1Trig
enumerator kINPUTMUX__Ctimer0Mat3ToCtimer1Trig
enumerator kINPUTMUX__Pwm0Sm0Muxtrig0ToCtimer1Trig
enumerator kINPUTMUX__Pwm0Sm1Muxtrig0ToCtimer1Trig
enumerator kINPUTMUX__Pwm0Sm2Muxtrig0ToCtimer1Trig
enumerator kINPUTMUX__Lpi2C0MasterendofpacketToCtimer1Trig
enumerator kINPUTMUX__Lpi2C0SlaveendofpacketToCtimer1Trig
enumerator kINPUTMUX__Lpspi0EndofframeToCtimer1Trig
enumerator kINPUTMUX__Lpspi0ReceiveddatawordToCtimer1Trig
enumerator kINPUTMUX__Lpuart0ReceiveddatawordToCtimer1Trig
enumerator kINPUTMUX__Lpuart0TransmitteddatawordToCtimer1Trig
enumerator kINPUTMUX__Lpuart0ReceivelineidleToCtimer1Trig
enumerator kINPUTMUX__Lpuart1ReceiveddatawordToCtimer1Trig
enumerator kINPUTMUX__Lpuart1TransmitteddatawordToCtimer1Trig
enumerator kINPUTMUX__Lpuart1ReceivelineidleToCtimer1Trig
enumerator kINPUTMUX__Lpuart2ReceiveddatawordToCtimer1Trig
enumerator kINPUTMUX__Lpuart2TransmitteddatawordToCtimer1Trig
enumerator kINPUTMUX__Lpuart2ReceivelineidleToCtimer1Trig
enumerator kINPUTMUX__Trigin0ToCtimer1Trig
enumerator kINPUTMUX__Trigin1ToCtimer1Trig
enumerator kINPUTMUX__Trigin2ToCtimer1Trig
enumerator kINPUTMUX__Trigin3ToCtimer1Trig
enumerator kINPUTMUX__Trigin4ToCtimer1Trig
enumerator kINPUTMUX__Trigin5ToCtimer1Trig
enumerator kINPUTMUX__Trigin6ToCtimer1Trig
enumerator kINPUTMUX__Trigin7ToCtimer1Trig
enumerator kINPUTMUX__Trigin8ToCtimer1Trig
enumerator kINPUTMUX__Trigin9ToCtimer1Trig
enumerator kINPUTMUX__Trigin10ToCtimer1Trig
enumerator kINPUTMUX__Trigin11ToCtimer1Trig

enumerator kINPUTMUX_Lpuart3ReceiveddatawordToCtimer1Trig
enumerator kINPUTMUX_Lpuart3TransmitteddatawordToCtimer1Trig
enumerator kINPUTMUX_Lpuart3ReceivelineidleToCtimer1Trig
enumerator kINPUTMUX_ClkinToFreqmeasref
enumerator kINPUTMUX_Froosc12MToFreqmeasref
enumerator kINPUTMUX_FrohfddivToFreqmeasref
enumerator kINPUTMUX_Osc32K1ToFreqmeasref
enumerator kINPUTMUX_Clk16K1ToFreqmeasref
enumerator kINPUTMUX_SlowclkToFreqmeasref
enumerator kINPUTMUX_Freqmeclkin0ToFreqmeasref
enumerator kINPUTMUX_Freqmeclkin1ToFreqmeasref
enumerator kINPUTMUX_Pwm0Sm0Muxtrig0ToFreqmeasref
enumerator kINPUTMUX_Pwm0Sm0Muxtrig1ToFreqmeasref
enumerator kINPUTMUX_Pwm0Sm1Muxtrig0ToFreqmeasref
enumerator kINPUTMUX_Pwm0Sm1Muxtrig1ToFreqmeasref
enumerator kINPUTMUX_Pwm0Sm2Muxtrig0ToFreqmeasref
enumerator kINPUTMUX_Pwm0Sm2Muxtrig1ToFreqmeasref
enumerator kINPUTMUX_ClkinToFreqmeastar
enumerator kINPUTMUX_Froosc12MToFreqmeastar
enumerator kINPUTMUX_FrohfddivToFreqmeastar
enumerator kINPUTMUX_Osc32K1ToFreqmeastar
enumerator kINPUTMUX_Clk16K1ToFreqmeastar
enumerator kINPUTMUX_SlowclkToFreqmeastar
enumerator kINPUTMUX_Freqmeclkin0ToFreqmeastar
enumerator kINPUTMUX_Freqmeclkin1ToFreqmeastar
enumerator kINPUTMUX_Pwm0Sm0Muxtrig0ToFreqmeastar
enumerator kINPUTMUX_Pwm0Sm0Muxtrig1ToFreqmeastar
enumerator kINPUTMUX_Pwm0Sm1Muxtrig0ToFreqmeastar
enumerator kINPUTMUX_Pwm0Sm1Muxtrig1ToFreqmeastar
enumerator kINPUTMUX_Pwm0Sm2Muxtrig0ToFreqmeastar
enumerator kINPUTMUX_Pwm0Sm2Muxtrig1ToFreqmeastar
enumerator kINPUTMUX_Ctimer0Mat0ToCmp0Trig
enumerator kINPUTMUX_Ctimer0Mat2ToCmp0Trig

enumerator kINPUTMUX__Ctimer1Mat0ToCmp0Trig
enumerator kINPUTMUX__Ctimer1Mat2ToCmp0Trig
enumerator kINPUTMUX__Lptmr0ToCmp0Trig
enumerator kINPUTMUX__Pwm0Sm0Muxtrig0ToCmp0Trig
enumerator kINPUTMUX__Pwm0Sm0Muxtrig1ToCmp0Trig
enumerator kINPUTMUX__Pwm0Sm1Muxtrig0ToCmp0Trig
enumerator kINPUTMUX__Pwm0Sm1Muxtrig1ToCmp0Trig
enumerator kINPUTMUX__Pwm0Sm2Muxtrig0ToCmp0Trig
enumerator kINPUTMUX__Pwm0Sm2Muxtrig1ToCmp0Trig
enumerator kINPUTMUX__Gpio0Pineventtrig0ToCmp0Trig
enumerator kINPUTMUX__Gpio1Pineventtrig0ToCmp0Trig
enumerator kINPUTMUX__Gpio2Pineventtrig0ToCmp0Trig
enumerator kINPUTMUX__Gpio3Pineventtrig0ToCmp0Trig
enumerator kINPUTMUX__WuuToCmp0Trig
enumerator kINPUTMUX__ArmtxevToAdc0Trig
enumerator kINPUTMUX__Cmp0OutToAdc0Trig
enumerator kINPUTMUX__Ctimer0Mat0ToAdc0Trig
enumerator kINPUTMUX__Ctimer0Mat1ToAdc0Trig
enumerator kINPUTMUX__Ctimer1Mat0ToAdc0Trig
enumerator kINPUTMUX__Ctimer1Mat1ToAdc0Trig
enumerator kINPUTMUX__Lptmr0ToAdc0Trig
enumerator kINPUTMUX__Pwm0Sm0Muxtrig0ToAdc0Trig
enumerator kINPUTMUX__Pwm0Sm0Muxtrig1ToAdc0Trig
enumerator kINPUTMUX__Pwm0Sm1Muxtrig0ToAdc0Trig
enumerator kINPUTMUX__Pwm0Sm1Muxtrig1ToAdc0Trig
enumerator kINPUTMUX__Pwm0Sm2Muxtrig0ToAdc0Trig
enumerator kINPUTMUX__Pwm0Sm2Muxtrig1ToAdc0Trig
enumerator kINPUTMUX__Gpio0Pineventtrig0ToAdc0Trig
enumerator kINPUTMUX__Gpio1Pineventtrig0ToAdc0Trig
enumerator kINPUTMUX__Gpio2Pineventtrig0ToAdc0Trig
enumerator kINPUTMUX__Gpio3Pineventtrig0ToAdc0Trig
enumerator kINPUTMUX__WuuToAdc0Trig
enumerator kINPUTMUX__ArmtxevToFlexpwm0Sm0Exta0

enumerator kINPUTMUX__Cmp0OutToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Ctimer0Mat2ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Ctimer0Mat3ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Ctimer1Mat2ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Ctimer1Mat3ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin0ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin1ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin2ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin3ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin4ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin5ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin6ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin7ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin8ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin9ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin10ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Trigin11ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Gpio0Pineventtrig0ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Gpio1Pineventtrig0ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Gpio2Pineventtrig0ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__Gpio3Pineventtrig0ToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__EccerrorinterruptToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__WwdtinterruptToFlexpwm0Sm0Ext0
enumerator kINPUTMUX__ArmtxevToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Cmp0OutToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Ctimer0Mat2ToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Ctimer0Mat3ToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Ctimer1Mat2ToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Ctimer1Mat3ToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Trigin0ToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Trigin1ToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Trigin2ToFlexpwm0Sm1Ext0
enumerator kINPUTMUX__Trigin3ToFlexpwm0Sm1Ext0

enumerator kINPUTMUX__Trigin4ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Trigin5ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Trigin6ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Trigin7ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Trigin8ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Trigin9ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Trigin10ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Trigin11ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Gpio0Pineventtrig0ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Gpio1Pineventtrig0ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Gpio2Pineventtrig0ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__Gpio3Pineventtrig0ToFlexpwm0Sm1Exta
enumerator kINPUTMUX__EccerrorinterruptToFlexpwm0Sm1Exta
enumerator kINPUTMUX__WwdtinterruptToFlexpwm0Sm1Exta
enumerator kINPUTMUX__ArmtxeVToFlexpwm0Sm2Exta
enumerator kINPUTMUX__Cmp0OutToFlexpwm0Sm2Exta
enumerator kINPUTMUX__Ctimer0Mat2ToFlexpwm0Sm2Exta
enumerator kINPUTMUX__Ctimer0Mat3ToFlexpwm0Sm2Exta
enumerator kINPUTMUX__Ctimer1Mat2ToFlexpwm0Sm2Exta
enumerator kINPUTMUX__Ctimer1Mat3ToFlexpwm0Sm2Exta
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__Trigin10ToFlexpwm0Sm2Exta
enumerator kINPUTMUX__Trigin11ToFlexpwm0Sm2Exta
enumerator kINPUTMUX__Gpio0Pineventtrig0ToFlexpwm0Sm2Exta

enumerator kINPUTMUX_Gpio1Pineventtrig0ToFlexpwm0Sm2Ext
enumerator kINPUTMUX_Gpio2Pineventtrig0ToFlexpwm0Sm2Ext
enumerator kINPUTMUX_Gpio3Pineventtrig0ToFlexpwm0Sm2Ext
enumerator kINPUTMUX_EccerrorinterruptToFlexpwm0Sm2Ext
enumerator kINPUTMUX_WwdtinterruptToFlexpwm0Sm2Ext
enumerator kINPUTMUX_ArmtxevToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Cmp0OutToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Ctimer0Mat2ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Ctimer0Mat3ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Ctimer1Mat2ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Ctimer1Mat3ToFlexpwm0Sm0Extsync
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_Trigin10ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Trigin11ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Gpio0Pineventtrig0ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Gpio1Pineventtrig0ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Gpio2Pineventtrig0ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_Gpio3Pineventtrig0ToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_EccerrorinterruptToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_WwdtinterruptToFlexpwm0Sm0Extsync
enumerator kINPUTMUX_ArmtxevToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Cmp0OutToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Ctimer0Mat2ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Ctimer0Mat3ToFlexpwm0Sm1Extsync

enumerator kINPUTMUX_Ctimer1Mat2ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Ctimer1Mat3ToFlexpwm0Sm1Extsync
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_Trigin10ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Trigin11ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Gpio0Pineventtrig0ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Gpio1Pineventtrig0ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Gpio2Pineventtrig0ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_Gpio3Pineventtrig0ToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_EccerrorinterruptToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_WwdtinterruptToFlexpwm0Sm1Extsync
enumerator kINPUTMUX_ArmtxeVToFlexpwm0Sm2Extsync
enumerator kINPUTMUX_Cmp0OutToFlexpwm0Sm2Extsync
enumerator kINPUTMUX_Ctimer0Mat2ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX_Ctimer0Mat3ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX_Ctimer1Mat2ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX_Ctimer1Mat3ToFlexpwm0Sm2Extsync
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__Trigin10ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__Trigin11ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__Gpio0Pineventtrig0ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__Gpio1Pineventtrig0ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__Gpio2Pineventtrig0ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__Gpio3Pineventtrig0ToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__EccerrorinterruptToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__WwdtinterruptToFlexpwm0Sm2Extsync
enumerator kINPUTMUX__ArmtxevToFlexpwm0Fault
enumerator kINPUTMUX__Cmp0OutToFlexpwm0Fault
enumerator kINPUTMUX__Ctimer0Mat2ToFlexpwm0Fault
enumerator kINPUTMUX__Ctimer0Mat3ToFlexpwm0Fault
enumerator kINPUTMUX__Ctimer1Mat2ToFlexpwm0Fault
enumerator kINPUTMUX__Ctimer1Mat3ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin0ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin1ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin2ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin3ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin4ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin5ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin6ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin7ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin8ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin9ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin10ToFlexpwm0Fault
enumerator kINPUTMUX__Trigin11ToFlexpwm0Fault
enumerator kINPUTMUX__Gpio0Pineventtrig0ToFlexpwm0Fault
enumerator kINPUTMUX__Gpio1Pineventtrig0ToFlexpwm0Fault
enumerator kINPUTMUX__Gpio2Pineventtrig0ToFlexpwm0Fault
enumerator kINPUTMUX__Gpio3Pineventtrig0ToFlexpwm0Fault

enumerator kINPUTMUX__EccerrorinterruptToFlexpwm0Fault
enumerator kINPUTMUX__WwdtinterruptToFlexpwm0Fault
enumerator kINPUTMUX__ArmtxeVToFlexpwm0Force
enumerator kINPUTMUX__Cmp0OutToFlexpwm0Force
enumerator kINPUTMUX__Ctimer0Mat2ToFlexpwm0Force
enumerator kINPUTMUX__Ctimer0Mat3ToFlexpwm0Force
enumerator kINPUTMUX__Ctimer1Mat2ToFlexpwm0Force
enumerator kINPUTMUX__Ctimer1Mat3ToFlexpwm0Force
enumerator kINPUTMUX__Trigin0ToFlexpwm0Force
enumerator kINPUTMUX__Trigin1ToFlexpwm0Force
enumerator kINPUTMUX__Trigin2ToFlexpwm0Force
enumerator kINPUTMUX__Trigin3ToFlexpwm0Force
enumerator kINPUTMUX__Trigin4ToFlexpwm0Force
enumerator kINPUTMUX__Trigin5ToFlexpwm0Force
enumerator kINPUTMUX__Trigin6ToFlexpwm0Force
enumerator kINPUTMUX__Trigin7ToFlexpwm0Force
enumerator kINPUTMUX__Trigin8ToFlexpwm0Force
enumerator kINPUTMUX__Trigin9ToFlexpwm0Force
enumerator kINPUTMUX__Trigin10ToFlexpwm0Force
enumerator kINPUTMUX__Trigin11ToFlexpwm0Force
enumerator kINPUTMUX__Gpio0Pineventtrig0ToFlexpwm0Force
enumerator kINPUTMUX__Gpio1Pineventtrig0ToFlexpwm0Force
enumerator kINPUTMUX__Gpio2Pineventtrig0ToFlexpwm0Force
enumerator kINPUTMUX__Gpio3Pineventtrig0ToFlexpwm0Force
enumerator kINPUTMUX__EccerrorinterruptToFlexpwm0Force
enumerator kINPUTMUX__WwdtinterruptToFlexpwm0Force
enumerator kINPUTMUX__Clk16K1ToPwm0Extclk
enumerator kINPUTMUX__ClkinToPwm0Extclk
enumerator kINPUTMUX__Trigin0ToPwm0Extclk
enumerator kINPUTMUX__Trigin7ToPwm0Extclk
enumerator kINPUTMUX__Cmp0OutToExttrig
enumerator kINPUTMUX__Lpuart0IppdolpuarttxdToExttrig
enumerator kINPUTMUX__Lpuart1IppdolpuarttxdToExttrig

enumerator kINPUTMUX__Lpuart2IppdolpuarttxdToExttrig
enumerator kINPUTMUX__Lpuart3IppdolpuarttxdToExttrig
enumerator kINPUTMUX__Rtc1HzclkToExttrig
enumerator kINPUTMUX__Cmp0OutToLpi2C0Trig
enumerator kINPUTMUX__Ctimer0Mat0ToLpi2C0Trig
enumerator kINPUTMUX__Ctimer0Mat1ToLpi2C0Trig
enumerator kINPUTMUX__Ctimer1Mat0ToLpi2C0Trig
enumerator kINPUTMUX__Ctimer1Mat1ToLpi2C0Trig
enumerator kINPUTMUX__Lptmr0ToLpi2C0Trig
enumerator kINPUTMUX__Trigin0ToLpi2C0Trig
enumerator kINPUTMUX__Trigin1ToLpi2C0Trig
enumerator kINPUTMUX__Trigin2ToLpi2C0Trig
enumerator kINPUTMUX__Trigin3ToLpi2C0Trig
enumerator kINPUTMUX__Trigin4ToLpi2C0Trig
enumerator kINPUTMUX__Trigin5ToLpi2C0Trig
enumerator kINPUTMUX__Trigin6ToLpi2C0Trig
enumerator kINPUTMUX__Trigin7ToLpi2C0Trig
enumerator kINPUTMUX__Gpio0Pineventtrig0ToLpi2C0Trig
enumerator kINPUTMUX__Gpio1Pineventtrig0ToLpi2C0Trig
enumerator kINPUTMUX__Gpio2Pineventtrig0ToLpi2C0Trig
enumerator kINPUTMUX__Gpio3Pineventtrig0ToLpi2C0Trig
enumerator kINPUTMUX__WuuToLpi2C0Trig
enumerator kINPUTMUX__Cmp0OutToLpspi0Trig
enumerator kINPUTMUX__Ctimer0Mat1ToLpspi0Trig
enumerator kINPUTMUX__Ctimer0Mat2ToLpspi0Trig
enumerator kINPUTMUX__Ctimer1Mat1ToLpspi0Trig
enumerator kINPUTMUX__Ctimer1Mat2ToLpspi0Trig
enumerator kINPUTMUX__Lptmr0ToLpspi0Trig
enumerator kINPUTMUX__Trigin0ToLpspi0Trig
enumerator kINPUTMUX__Trigin1ToLpspi0Trig
enumerator kINPUTMUX__Trigin2ToLpspi0Trig
enumerator kINPUTMUX__Trigin3ToLpspi0Trig
enumerator kINPUTMUX__Trigin4ToLpspi0Trig

enumerator kINPUTMUX__Trigin5ToLpspi0Trig
enumerator kINPUTMUX__Trigin6ToLpspi0Trig
enumerator kINPUTMUX__Trigin7ToLpspi0Trig
enumerator kINPUTMUX__Gpio0Pineventtrig0ToLpspi0Trig
enumerator kINPUTMUX__Gpio1Pineventtrig0ToLpspi0Trig
enumerator kINPUTMUX__Gpio2Pineventtrig0ToLpspi0Trig
enumerator kINPUTMUX__Gpio3Pineventtrig0ToLpspi0Trig
enumerator kINPUTMUX__WuuToLpspi0Trig
enumerator kINPUTMUX__Cmp0OutToLpuart0
enumerator kINPUTMUX__Ctimer0Mat2ToLpuart0
enumerator kINPUTMUX__Ctimer0Mat3ToLpuart0
enumerator kINPUTMUX__Ctimer1Mat2ToLpuart0
enumerator kINPUTMUX__Ctimer1Mat3ToLpuart0
enumerator kINPUTMUX__Lptmr0ToLpuart0
enumerator kINPUTMUX__Trigin0ToLpuart0
enumerator kINPUTMUX__Trigin1ToLpuart0
enumerator kINPUTMUX__Trigin2ToLpuart0
enumerator kINPUTMUX__Trigin3ToLpuart0
enumerator kINPUTMUX__Trigin4ToLpuart0
enumerator kINPUTMUX__Trigin5ToLpuart0
enumerator kINPUTMUX__Trigin6ToLpuart0
enumerator kINPUTMUX__Trigin7ToLpuart0
enumerator kINPUTMUX__Trigin8ToLpuart0
enumerator kINPUTMUX__Trigin9ToLpuart0
enumerator kINPUTMUX__Trigin10ToLpuart0
enumerator kINPUTMUX__Trigin11ToLpuart0
enumerator kINPUTMUX__Gpio0Pineventtrig0ToLpuart0
enumerator kINPUTMUX__Gpio1Pineventtrig0ToLpuart0
enumerator kINPUTMUX__Gpio2Pineventtrig0ToLpuart0
enumerator kINPUTMUX__Gpio3Pineventtrig0ToLpuart0
enumerator kINPUTMUX__WuuToLpuart0
enumerator kINPUTMUX__Cmp0OutToLpuart1
enumerator kINPUTMUX__Ctimer0Mat2ToLpuart1

enumerator kINPUTMUX_Ctimer0Mat3ToLpuart1
enumerator kINPUTMUX_Ctimer1Mat2ToLpuart1
enumerator kINPUTMUX_Ctimer1Mat3ToLpuart1
enumerator kINPUTMUX_Lptmr0ToLpuart1
enumerator kINPUTMUX_Trigin0ToLpuart1
enumerator kINPUTMUX_Trigin1ToLpuart1
enumerator kINPUTMUX_Trigin2ToLpuart1
enumerator kINPUTMUX_Trigin3ToLpuart1
enumerator kINPUTMUX_Trigin4ToLpuart1
enumerator kINPUTMUX_Trigin5ToLpuart1
enumerator kINPUTMUX_Trigin6ToLpuart1
enumerator kINPUTMUX_Trigin7ToLpuart1
enumerator kINPUTMUX_Trigin8ToLpuart1
enumerator kINPUTMUX_Trigin9ToLpuart1
enumerator kINPUTMUX_Trigin10ToLpuart1
enumerator kINPUTMUX_Trigin11ToLpuart1
enumerator kINPUTMUX_Gpio0Pineventtrig0ToLpuart1
enumerator kINPUTMUX_Gpio1Pineventtrig0ToLpuart1
enumerator kINPUTMUX_Gpio2Pineventtrig0ToLpuart1
enumerator kINPUTMUX_Gpio3Pineventtrig0ToLpuart1
enumerator kINPUTMUX_WuuToLpuart1
enumerator kINPUTMUX_Cmp0OutToLpuart2
enumerator kINPUTMUX_Ctimer0Mat2ToLpuart2
enumerator kINPUTMUX_Ctimer0Mat3ToLpuart2
enumerator kINPUTMUX_Ctimer1Mat2ToLpuart2
enumerator kINPUTMUX_Ctimer1Mat3ToLpuart2
enumerator kINPUTMUX_Lptmr0ToLpuart2
enumerator kINPUTMUX_Trigin0ToLpuart2
enumerator kINPUTMUX_Trigin1ToLpuart2
enumerator kINPUTMUX_Trigin2ToLpuart2
enumerator kINPUTMUX_Trigin3ToLpuart2
enumerator kINPUTMUX_Trigin4ToLpuart2
enumerator kINPUTMUX_Trigin5ToLpuart2

enumerator kINPUTMUX__Trigin6ToLpuart2
enumerator kINPUTMUX__Trigin7ToLpuart2
enumerator kINPUTMUX__Trigin8ToLpuart2
enumerator kINPUTMUX__Trigin9ToLpuart2
enumerator kINPUTMUX__Trigin10ToLpuart2
enumerator kINPUTMUX__Trigin11ToLpuart2
enumerator kINPUTMUX__Gpio0Pineventtrig0ToLpuart2
enumerator kINPUTMUX__Gpio1Pineventtrig0ToLpuart2
enumerator kINPUTMUX__Gpio2Pineventtrig0ToLpuart2
enumerator kINPUTMUX__Gpio3Pineventtrig0ToLpuart2
enumerator kINPUTMUX__WuuToLpuart2
enumerator kINPUTMUX__Cmp0OutToLpuart3
enumerator kINPUTMUX__Ctimer0Mat2ToLpuart3
enumerator kINPUTMUX__Ctimer0Mat3ToLpuart3
enumerator kINPUTMUX__Ctimer1Mat2ToLpuart3
enumerator kINPUTMUX__Ctimer1Mat3ToLpuart3
enumerator kINPUTMUX__Lptmr0ToLpuart3
enumerator kINPUTMUX__Trigin0ToLpuart3
enumerator kINPUTMUX__Trigin1ToLpuart3
enumerator kINPUTMUX__Trigin2ToLpuart3
enumerator kINPUTMUX__Trigin3ToLpuart3
enumerator kINPUTMUX__Trigin4ToLpuart3
enumerator kINPUTMUX__Trigin5ToLpuart3
enumerator kINPUTMUX__Trigin6ToLpuart3
enumerator kINPUTMUX__Trigin7ToLpuart3
enumerator kINPUTMUX__Trigin8ToLpuart3
enumerator kINPUTMUX__Trigin9ToLpuart3
enumerator kINPUTMUX__Trigin10ToLpuart3
enumerator kINPUTMUX__Trigin11ToLpuart3
enumerator kINPUTMUX__Gpio0Pineventtrig0ToLpuart3
enumerator kINPUTMUX__Gpio1Pineventtrig0ToLpuart3
enumerator kINPUTMUX__Gpio2Pineventtrig0ToLpuart3
enumerator kINPUTMUX__Gpio3Pineventtrig0ToLpuart3


```
    enumerator kINPUTMUX_WuuToLpuart3
typedef enum _inputmux_index_t inputmux_index_t
typedef enum _inputmux_connection_t inputmux_connection_t
    INPUTMUX connections type.
CTIMER0CAP_REG
    Periphinmux IDs.
CTIMER0TRIG_REG
CTIMER1CAP_REG
CTIMER1TRIG_REG
FREQMEAS_REF_REG
FREQMEAS_TAR_REG
CMP0_TRIG_REG
ADC0_TRIG_REG
FlexPWM0_SM0_EXT_A0_REG
FlexPWM0_SM0_EXTSYNC_REG
FlexPWM0_SM1_EXT_A0_REG
FlexPWM0_SM1_EXTSYNC_REG
FlexPWM0_SM2_EXT_A0_REG
FlexPWM0_SM2_EXTSYNC_REG
FlexPWM0_FAULT_REG
FlexPWM0_FORCE_REG
PWM0_EXT_CLK_REG
EXT_TRIG_REG
LPI2C0_TRIG_REG
LPSPI0_TRIG_REG
LPUART0_REG
LPUART1_REG
LPUART2_REG
LPUART3_REG
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__Deinit(void *base)`

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

Parameters

- `base` – Base address of the INPUTMUX peripheral.

Return values

None. –

2.16 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.

MCUX_MASK_INVERT_8(*mask*)

8-bit mask inversion.

MCUX_MASK_INVERT_16(*mask*)

16-bit mask inversion.

MCUX_MASK_INVERT_32(*mask*)

32-bit mask inversion for completeness.

MCUX_REG_WRITE8(*reg*, *value*)

8-bit register write macro

MCUX_REG_WRITE16(*reg*, *value*)

16-bit register write macro

MCUX_REG_WRITE32(*reg*, *value*)

32-bit register write macro

MCUX_REG_READ8(*reg*)

8-bit register read macro

MCUX_REG_READ16(*reg*)

16-bit register read macro

MCUX_REG_READ32(*reg*)

32-bit register read macro

MCUX_REG_BIT_SET8(*reg*, *mask*)

8-bit register bit set macro

MCUX_REG_BIT_SET16(reg, mask)

16-bit register bit set macro

MCUX_REG_BIT_SET32(reg, mask)

32-bit register bit set macro

MCUX_REG_BIT_CLEAR8(reg, mask)

8-bit register bit clear macro

MCUX_REG_BIT_CLEAR16(reg, mask)

16-bit register bit clear macro

MCUX_REG_BIT_CLEAR32(reg, mask)

32-bit register bit clear macro

MCUX_REG_BIT_GET8(reg, mask)

8-bit register bit get macro

MCUX_REG_BIT_GET16(reg, mask)

16-bit register bit get macro

MCUX_REG_BIT_GET32(reg, mask)

32-bit register bit get macro

MCUX_REG_MODIFY8(reg, mask, value)

32-bit register read-modify-write macro

MCUX_REG_MODIFY16(reg, mask, value)

16-bit register read-modify-write macro

MCUX_REG_MODIFY32(reg, mask, value)

32-bit register read-modify-write macro

SDK_ATOMIC_LOCAL_ADD(addr, val)

Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)

Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)

Set the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)

Clear the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specified by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specified by *clearBits* and set the bits specified by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value (rounded up)

SDK_SIZEALIGN_UP(var, alignbytes)

Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK_SIZEALIGN

SDK_SIZEALIGN_DOWN(var, alignbytes)

Macro to change a value to a given size aligned value (rounded down)

SDK_IS_ALIGNED(var, alignbytes)

Macro to check if a value is aligned to a given size

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*.

MCUX_RAMFUNC

Function attribute to place function in RAM. For example, to place function `my_func` in ram, use like:

```
MCUX_RAMFUNC my_func
```

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_QUEUE DSPI
Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.

enumerator `kStatusGroup_CLIF`
Group number for CLIF status codes.

enumerator `kStatusGroup_BMA`
Group number for BMA status codes.

enumerator `kStatusGroup_NETC`
Group number for NETC status codes.

enumerator `kStatusGroup_ELE`
Group number for ELE status codes.

enumerator `kStatusGroup_GLIKEY`
Group number for GLIKEY status codes.

enumerator `kStatusGroup_AON_POWER`
Group number for AON_POWER status codes.

enumerator `kStatusGroup_AON_COMMON`
Group number for AON_COMMON status codes.

enumerator `kStatusGroup_ENDAT3`
Group number for ENDAT3 status codes.

enumerator `kStatusGroup_HIPERFACE`
Group number for HIPERFACE status codes.

enumerator `kStatusGroup_NPX`
Group number for NPX status codes.

enumerator `kStatusGroup_ELA_CSEC`
Group number for ELA_CSEC status codes.

enumerator `kStatusGroup_FLEXIO_T_FORMAT`
Group number for T-format status codes.

enumerator `kStatusGroup_FLEXIO_A_FORMAT`
Group number for A-format status codes.

enumerator `kStatusGroup_LPC_QSPI`
Group number for LPC QSPI 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.

2.17 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum _lpadc_status_flags

Define hardware flags of the module.

Values:

enumerator kLPADC_ResultFIFO0OverflowFlag

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC_ResultFIFO0ReadyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC_TriggerExceptionFlag

Indicates that a trigger exception event has occurred.

enumerator kLPADC_TriggerCompletionFlag

Indicates that a trigger completion event has occurred.

enumerator kLPADC_CalibrationReadyFlag

Indicates that the calibration process is done.

enumerator kLPADC_ActiveFlag

Indicates that the ADC is in active state.

enumerator kLPADC_ResultFIFOOverflowFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowFlag as instead.

enumerator kLPADC_ResultFIFOReadyFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0ReadyFlag as instead.

enum _lpadc_interrupt_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowInterruptEnable as instead.

enumerator kLPADC_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_FIFO0WatermarkInterruptEnable as instead.

enumerator kLPADC_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kLPADC_Trigger2CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 2 completion.

enumerator kLPADC_Trigger3CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 3 completion.

enumerator kLPADC_Trigger4CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 4 completion.

enumerator kLPADC_Trigger5CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 5 completion.

enumerator kLPADC_Trigger6CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 6 completion.

enumerator kLPADC_Trigger7CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 7 completion.

enumerator kLPADC_Trigger8CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 8 completion.

enumerator kLPADC_Trigger9CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 9 completion.

enumerator kLPADC_Trigger10CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 10 completion.

enumerator kLPADC_Trigger11CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 11 completion.

enumerator kLPADC_Trigger12CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 12 completion.

enumerator kLPADC_Trigger13CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 13 completion.

enumerator kLPADC_Trigger14CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 14 completion.

enumerator kLPADC_Trigger15CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 15 completion.

enum _lpadc_trigger_status_flags

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_Trigger0InterruptedFlag

Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC_Trigger1InterruptedFlag

Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC_Trigger2InterruptedFlag

Trigger 2 is interrupted by a high priority exception.

enumerator kLPADC_Trigger3InterruptedFlag

Trigger 3 is interrupted by a high priority exception.

enumerator kLPADC_Trigger4InterruptedFlag

Trigger 4 is interrupted by a high priority exception.

enumerator kLPADC_Trigger5InterruptedFlag

Trigger 5 is interrupted by a high priority exception.

enumerator kLPADC_Trigger6InterruptedFlag

Trigger 6 is interrupted by a high priority exception.

enumerator kLPADC_Trigger7InterruptedFlag

Trigger 7 is interrupted by a high priority exception.

enumerator kLPADC_Trigger8InterruptedFlag

Trigger 8 is interrupted by a high priority exception.

enumerator kLPADC_Trigger9InterruptedFlag

Trigger 9 is interrupted by a high priority exception.

enumerator kLPADC_Trigger10InterruptedFlag

Trigger 10 is interrupted by a high priority exception.

enumerator kLPADC_Trigger11InterruptedFlag

Trigger 11 is interrupted by a high priority exception.

enumerator kLPADC_Trigger12InterruptedFlag

Trigger 12 is interrupted by a high priority exception.

enumerator kLPADC_Trigger13InterruptedFlag

Trigger 13 is interrupted by a high priority exception.

enumerator kLPADC_Trigger14InterruptedFlag

Trigger 14 is interrupted by a high priority exception.

enumerator kLPADC_Trigger15InterruptedFlag

Trigger 15 is interrupted by a high priority exception.

enumerator kLPADC_Trigger0CompletedFlag

Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC_Trigger1CompletedFlag

Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kLPADC_Trigger2CompletedFlag

Trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kLPADC_Trigger3CompletedFlag

Trigger 3 is completed and trigger 3 has enabled completion interrupts.

enumerator kLPADC_Trigger4CompletedFlag

Trigger 4 is completed and trigger 4 has enabled completion interrupts.

enumerator kLPADC_Trigger5CompletedFlag

Trigger 5 is completed and trigger 5 has enabled completion interrupts.

enumerator kLPADC_Trigger6CompletedFlag

Trigger 6 is completed and trigger 6 has enabled completion interrupts.

enumerator kLPADC_Trigger7CompletedFlag

Trigger 7 is completed and trigger 7 has enabled completion interrupts.

enumerator kLPADC_Trigger8CompletedFlag

Trigger 8 is completed and trigger 8 has enabled completion interrupts.

enumerator kLPADC_Trigger9CompletedFlag

Trigger 9 is completed and trigger 9 has enabled completion interrupts.

enumerator kLPADC_Trigger10CompletedFlag

Trigger 10 is completed and trigger 10 has enabled completion interrupts.

enumerator kLPADC_Trigger11CompletedFlag

Trigger 11 is completed and trigger 11 has enabled completion interrupts.

enumerator kLPADC_Trigger12CompletedFlag

Trigger 12 is completed and trigger 12 has enabled completion interrupts.

enumerator kLPADC_Trigger13CompletedFlag

Trigger 13 is completed and trigger 13 has enabled completion interrupts.

enumerator kLPADC_Trigger14CompletedFlag

Trigger 14 is completed and trigger 14 has enabled completion interrupts.

enumerator kLPADC_Trigger15CompletedFlag

Trigger 15 is completed and trigger 15 has enabled completion interrupts.

enum _lpadc_sample_scale_mode

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Values:

enumerator kLPADC_SamplePartScale

Use divided input voltage signal. (For scale select, please refer to the reference manual).

enumerator kLPADC_SampleFullScale

Full scale (Factor of 1).

enum __lpadc_sample_channel_mode

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Values:

enumerator kLPADC_SampleChannelSingleEndSideA

Single-end mode, only A-side channel is converted.

enumerator kLPADC_SampleChannelSingleEndSideB

Single-end mode, only B-side channel is converted.

enumerator kLPADC_SampleChannelDiffBothSideAB

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDiffBothSideBA

Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC_SampleChannelDiffBothSide

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDualSingleEndBothSide

Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum __lpadc_hardware_average_mode

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Values:

enumerator kLPADC_HardwareAverageCount1

Single conversion.

enumerator kLPADC_HardwareAverageCount2

2 conversions averaged.

enumerator kLPADC_HardwareAverageCount4

4 conversions averaged.

enumerator kLPADC_HardwareAverageCount8

8 conversions averaged.

enumerator kLPADC_HardwareAverageCount16

16 conversions averaged.

enumerator kLPADC_HardwareAverageCount32

32 conversions averaged.

enumerator kLPADC_HardwareAverageCount64
64 conversions averaged.

enumerator kLPADC_HardwareAverageCount128
128 conversions averaged.

enum _lpadc_sample_time_mode

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Values:

enumerator kLPADC_SampleTimeADCK3
3 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK5
5 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK7
7 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK11
11 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK19
19 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK35
35 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK67
69 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK131
131 ADCK cycles total sample time.

enum _lpadc_hardware_compare_mode

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Values:

enumerator kLPADC_HardwareCompareDisabled
Compare disabled.

enumerator kLPADC_HardwareCompareStoreOnTrue
Compare enabled. Store on true.

enumerator kLPADC_HardwareCompareRepeatUntilTrue
Compare enabled. Repeat channel acquisition until true.

enum _lpadc_conversion_resolution_mode

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to `lpadc_sample_channel_mode_t`

Values:

enumerator kLPADC_ConversionResolutionStandard

Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator kLPADC_ConversionResolutionHigh

High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum _lpadc_conversion_average_mode

Define enumeration of conversion averages mode.

Configure the conversion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

Values:

enumerator kLPADC_ConversionAverage1

Single conversion.

enumerator kLPADC_ConversionAverage2

2 conversions averaged.

enumerator kLPADC_ConversionAverage4

4 conversions averaged.

enumerator kLPADC_ConversionAverage8

8 conversions averaged.

enumerator kLPADC_ConversionAverage16

16 conversions averaged.

enumerator kLPADC_ConversionAverage32

32 conversions averaged.

enumerator kLPADC_ConversionAverage64

64 conversions averaged.

enumerator kLPADC_ConversionAverage128

128 conversions averaged.

enumerator kLPADC_ConversionAverageMax

enum _lpadc_reference_voltage_mode

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

Values:

enumerator kLPADC_ReferenceVoltageAlt1

Option 1 setting.

enumerator kLPADC_ReferenceVoltageAlt2

Option 2 setting.

enumerator kLPADC_ReferenceVoltageAlt3

Option 3 setting.

enum _lpadc_power_level_mode

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Values:

enumerator kLPADC_PowerLevelAlt1

Lowest power setting.

enumerator kLPADC_PowerLevelAlt2

Next lowest power setting.

enumerator kLPADC_PowerLevelAlt3

...

enumerator kLPADC_PowerLevelAlt4

Highest power setting.

enum _lpadc_offset_calibration_mode

Define enumeration of offset calibration mode.

Values:

enumerator kLPADC_OffsetCalibration12bitMode

12 bit offset calibration mode.

enumerator kLPADC_OffsetCalibration16bitMode

16 bit offset calibration mode.

enum _lpadc_trigger_priority_policy

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

Values:

enumerator kLPADC_ConvPreemptImmediatelyNotAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator kLPADC_ConvPreemptSoftlyNotAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoRestarted

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptSoftlyAutoRestarted

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator kLPADC_ConvPreemptSoftlyAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator kLPADC_TriggerPriorityPreemptImmediately

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityPreemptSoftly

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityExceptionDisabled

High priority trigger exception disabled.

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;
```

(continues on next page)

(continued from previous page)

```

config->powerLevelMode      = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause      = false;
config->convPauseDelay       = 0U;
config->FIFOWatermark        = 0U;

```

Parameters

- config – Pointer to configuration structure.

void LPADC_Deinit(ADC_Type *base)

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_Enable(ADC_Type *base, bool enable)

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC_DoResetFIFO(ADC_Type *base)

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_DoResetConfig(ADC_Type *base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns

status flags' mask. See to `_lpadc_status_flags`.

static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clear status flags.

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to `_lpadc_status_flags`.

static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Parameters

- base – LPADC peripheral base address.

Returns

The OR'ed value of `_lpadc_trigger_status_flags`.

```
static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)
```

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of `_lpadc_trigger_status_flags`.

```
static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)
```

Enable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

```
static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)
```

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to `_lpadc_interrupt_enable`.

```
static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)
```

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

```
static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)
```

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns

The count of result kept in conversion FIFO.

```
bool LPADC_GetConvResult(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns

Status whether FIFO entry is valid.

```
void LPADC_GetConvResultBlocking(ADC_Type *base, lpadc_conv_result_t *result)
```

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

```
void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const
                                lpadc_conv_trigger_config_t *config)
```

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to `lpadc_conv_trigger_config_t`.

```
void LPADC_GetDefaultConvTriggerConfig(lpadc_conv_trigger_config_t *config)
```

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId    = 0U;
config->delayPower         = 0U;
config->priority            = 0U;
config->channelAFIFOSelect  = 0U;
config->channelBFIFOSelect  = 0U;
config->enableHardwareTrigger = false;
```

Parameters

- config – Pointer to configuration structure.

```
static inline void LPADC_DoSoftwareTrigger(ADC_Type *base, uint32_t triggerIdMask)
```

Do software trigger to conversion command.

Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

```
static inline void LPADC_EnableHardwareTriggerCommandSelection(ADC_Type *base, uint32_t
                                                                triggerId, bool enable)
```

Enable hardware trigger command selection.

This function will use the hardware trigger command from ADC_ETC. The trigger command is then defined by ADC hardware trigger command selection field in ADC_ETC->TRIGx_CHAINy_z_n[CSEL].

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.

- enable – True to enable or false to disable.

```
void LPADC_SetConvCommandConfig(ADC_Type *base, uint32_t commandId, const  
                                lpadc_conv_command_config_t *config)
```

Configure conversion command.

Note: The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

Parameters

- base – LPADC peripheral base address.
- commandId – ID for command in command buffer. Typically, the available value range is 1 - 15.
- config – Pointer to configuration structure. See to lpadc_conv_command_config_t.

```
void LPADC_GetDefaultConvCommandConfig(lpadc_conv_command_config_t *config)
```

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode      = kLPADC_SampleFullScale;  
config->channelBScaleMode    = kLPADC_SampleFullScale;  
config->sampleChannelMode    = kLPADC_SampleChannelSingleEndSideA;  
config->channelNumber        = 0U;  
config->channelBNumber       = 0U;  
config->chainedNextCommandNumber = 0U;  
config->enableAutoChannelIncrement = false;  
config->loopCount             = 0U;  
config->hardwareAverageMode    = kLPADC_HardwareAverageCount1;  
config->sampleTimeMode        = kLPADC_SampleTimeADCK3;  
config->hardwareCompareMode    = kLPADC_HardwareCompareDisabled;  
config->hardwareCompareValueHigh = 0U;  
config->hardwareCompareValueLow  = 0U;  
config->conversionResolutionMode = kLPADC_ConversionResolutionStandard;  
config->enableWaitTrigger       = false;  
config->enableChannelB         = false;
```

Parameters

- config – Pointer to configuration structure.

```
void LPADC_EnableCalibration(ADC_Type *base, bool enable)
```

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6- bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

Parameters

- base – LPADC peripheral base address.
- value – Setting offset value.

status_t LPADC_DoAutoCalibration(ADC_Type *base)

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including:

- LPADC_EnableCalibration(...)
- LPADC_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*.

static inline void LPADC_EnableJustifiedLeft(ADC_Type *base, bool enable)

Enable/disable left-justify format in 12-bit single-end mode.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable left-justify format in 12-bit single-end mode:
 - **true** Enable left-justify format in 12-bit single-end mode;
 - **false** Disable left-justify format in 12-bit single-end mode.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.9.5.

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.

2.18 Lpc_freqme

void FREQME_Init(FREQME_Type *base, const *freq_measure_config_t* *config)

Initialize freqme module, set operate mode, operate mode attribute and initialize measurement cycle.

Parameters

- base – FREQME peripheral base address.
- config – The pointer to module basic configuration, please refer to *freq_measure_config_t*.

void FREQME_GetDefaultConfig(*freq_measure_config_t* *config)

Get default configuration.

```
config->operateMode = kFREQME_FreqMeasurementMode;
config->operateModeAttribute.refClkScaleFactor = 0U;
config->enableContinuousMode = false;
config->startMeasurement = false;
```

Parameters

- config – The pointer to module basic configuration, please refer to *freq_measure_config_t*.

static inline void FREQME_StartMeasurementCycle(FREQME_Type *base)

Start frequency or pulse width measurement process.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_TerminateMeasurementCycle(FREQME_Type *base)

Force the termination of any measurement cycle currently in progress and resets RESULT or just reset RESULT if the module in idle state.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_EnableContinuousMode(FREQME_Type *base, bool enable)

Enable/disable Continuous mode.

Parameters

- base – FREQME peripheral base address.
- enable – Used to enable/disable continuous mode,
 - **true** Enable Continuous mode.
 - **false** Disable Continuous mode.

static inline bool FREQME_CheckContinuousMode(FREQME_Type *base)

Check whether continuous mode is enabled.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Continuous mode is enabled, the measurement is performed continuously.
- False – Continuous mode is disabled.

static inline void FREQME_SetOperateMode(FREQME_Type *base, *freqme_operate_mode_t* operateMode)

Set operate mode of freqme module.

Parameters

- base – FREQME peripheral base address.
- operateMode – The operate mode to be set, please refer to *freqme_operate_mode_t*.

static inline bool FREQME_CheckOperateMode(FREQME_Type *base)

Check module's operate mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Pulse width measurement mode.
- False – Frequency measurement mode.

static inline void FREQME_SetMinExpectedValue(FREQME_Type *base, uint32_t minValue)

Set the minimum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- minValue – The minimum value to set, please note that this value is 31 bits width.

```
static inline void FREQME_SetMaxExpectedValue(FREQME_Type *base, uint32_t maxValue)
```

Set the maximum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- maxValue – The maximum value to set, please note that this value is 31 bits width.

```
uint32_t FREQME_CalculateTargetClkFreq(FREQME_Type *base, uint32_t refClkFrequency)
```

Calculate the frequency of selected target clock□

Note: The formula: $F_{target} = (RESULT - 2) * F_{reference} / 2^{REF\_SCALE}$.

Note: This function only useful when the operate mode is selected as frequency measurement mode.

Parameters

- base – FREQME peripheral base address.
- refClkFrequency – The frequency of reference clock.

Returns

The frequency of target clock the unit is Hz, if the output result is 0, please check the module's operate mode.

```
static inline uint8_t FREQME_GetReferenceClkScaleValue(FREQME_Type *base)
```

Get reference clock scaling factor.

Parameters

- base – FREQME peripheral base address.

Returns

Reference clock scaling factor, the reference count cycle is 2^{ref_scale} .

```
static inline void FREQME_SetPulsePolarity(FREQME_Type *base, freqme_pulse_polarity_t pulsePolarity)
```

Set pulse polarity when operate mode is selected as Pulse Width Measurement mode.

Parameters

- base – FREQME peripheral base address.
- pulsePolarity – The pulse polarity to be set, please refer to `freqme_pulse_polarity_t`.

```
static inline bool FREQME_CheckPulsePolarity(FREQME_Type *base)
```

Check pulse polarity when the operate mode is selected as pulse width measurement mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Low period.
- False – High period.

```
static inline uint32_t FREQME_GetMeasurementResult(FREQME_Type *base)
```

Get measurement result.

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 2.1.4.

```
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 **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

typedef struct *_freq_measure_config* freq_measure_config_t

The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

union *_freqme_mode_attribute*

#include <fsl_freqme.h> The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

Public Members

uint8_t refClkScaleFactor

Only useful in frequency measurement operate mode, used to set the reference clock counter scaling factor.

freqme_pulse_polarity_t pulsePolarity

Only Useful in pulse width measurement operate mode, used to set period polarity.

struct *_freq_measure_config*

#include <fsl_freqme.h> The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

Public Members

freqme_operate_mode_t operateMode

Select operate mode, please refer to *freqme_operate_mode_t*.

freqme_mode_attribute_t operateModeAttribute

Used to set the attribute of the selected operate mode, if the operate mode is selected as *kFREQME_FreqMeasurementMode* set *freqme_mode_attribute_t::refClkScaleFactor*, if operate mode is selected as *kFREOME_PulseWidthMeasurementMode*, please set *freqme_mode_attribute_t::pulsePolarity*.

bool enableContinuousMode

Enable/disable continuous mode, if continuous mode is enable, the measurement is performed continuously and the result for the last completed measurement is available in the result register.

2.19 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-0xFFFFFFFFFUL.

```
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 vaule 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.

enum `_lpcmp_plus_input_src`

LPCMP plus input source.

Values:

enumerator `kLPCMP_PlusInputSrcDac`

LPCMP plus input source from the internal 8-bit DAC output.

enumerator `kLPCMP_PlusInputSrcMux`

LPCMP plus input source from the analog 8-1 mux.

enum `_lpcmp_minus_input_src`

LPCMP minus input source.

Values:

enumerator `kLPCMP_MinusInputSrcDac`

LPCMP minus input source from the internal 8-bit DAC output.

enumerator `kLPCMP_MinusInputSrcMux`

LPCMP minus input source from the analog 8-1 mux.

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 enum *_lpcmp_plus_input_src* lpcmp_plus_input_src_t

LPCMP plus input source.

typedef enum *_lpcmp_minus_input_src* lpcmp_minus_input_src_t

LPCMP minus input source.

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 enableCmpToDacLink

Controls the link from the CMP enable to the DAC enable.

bool enableOutputPin

Decide whether to enable the comparator is available in selected pin.

bool useUnfilteredOutput

Decide whether to use unfiltered output.

bool enableInvertOutput

Decide whether to inverts the comparator output.

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.

lpcmp_plus_input_src_t plusInputSrc

Select LPCMP plus input source.

lpcmp_minus_input_src_t minusInputSrc

Select LPCMP minus input 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.

2.20 LPI2C: Low Power Inter-Integrated Circuit Driver

`void LPI2C_DriverIRQHandler(uint32_t instance)`

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

Parameters

- `instance` – LPI2C instance.

`FSL_LPI2C_DRIVER_VERSION`

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator `kStatus_LPI2C_Busy`

The master is already performing a transfer.

enumerator `kStatus_LPI2C_Idle`

The slave driver is idle.

enumerator `kStatus_LPI2C_Nak`

The slave device sent a NAK in response to a byte.

enumerator `kStatus_LPI2C_FifoError`

FIFO under run or overrun.

enumerator `kStatus_LPI2C_BitError`

Transferred bit was not seen on the bus.

enumerator `kStatus_LPI2C_ArbitrationLost`

Arbitration lost error.

enumerator `kStatus_LPI2C_PinLowTimeout`

SCL or SDA were held low longer than the timeout.

enumerator `kStatus_LPI2C_NoTransferInProgress`

Attempt to abort a transfer when one is not in progress.

enumerator `kStatus_LPI2C_DmaRequestFail`

DMA request failed.

enumerator `kStatus_LPI2C_Timeout`

Timeout polling status flags.

`IRQn_Type` `const kLpi2cMasterIrqs[]`

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

`IRQn_Type` `const kLpi2cSlaveIrqs[]`

`lpi2c_master_isr_t` `s_lpi2cMasterIsr`

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

`void` `*s_lpi2cMasterHandle[]`

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

`uint32_t` `LPI2C_GetInstance(LPI2C_Type *base)`

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

LPI2C instance number starting from 0.

`I2C_RETRY_TIMES`

Retry times for waiting flag.

2.21 LPI2C Master Driver

`void` `LPI2C_MasterGetDefaultConfig(lpi2c_master_config_t *masterConfig)`

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```

masterConfig->enableMaster      = true;
masterConfig->debugEnable       = false;
masterConfig->ignoreAck         = false;
masterConfig->pinConfig          = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz       = 100000U;
masterConfig->busIdleTimeout_ns  = 0;
masterConfig->pinLowTimeout_ns   = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable = false;
masterConfig->hostRequest.source  = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;

```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `lpi2c_master_config_t`.

```
void LPI2C_MasterInit(LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t
    sourceClock_Hz)
```

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `LPI2C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void LPI2C_MasterDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

```
void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const lpi2c_data_match_config_t
    *matchConfig)
```

Configures LPI2C master data match feature.

Parameters

- `base` – The LPI2C peripheral base address.
- `matchConfig` – Settings for the data match feature.

```
status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)
```

Convert provided flags to status code, and clear any errors if present.

Parameters

- `base` – The LPI2C peripheral base address.
- `status` – Current status flags value that will be checked.

Return values

- `kStatus__Success` –
- `kStatus__LPI2C__PinLowTimeout` –
- `kStatus__LPI2C__ArbitrationLost` –
- `kStatus__LPI2C__Nak` –
- `kStatus__LPI2C__FifoError` –

`status_t` `LPI2C__CheckForBusyBus(LPI2C_Type *base)`

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `kStatus__Success` –
- `kStatus__LPI2C__Busy` –

`static inline void` `LPI2C__MasterReset(LPI2C_Type *base)`

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

`static inline void` `LPI2C__MasterEnable(LPI2C_Type *base, bool enable)`

Enables or disables the LPI2C module as master.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as master.

`static inline uint32_t` `LPI2C__MasterGetStatusFlags(LPI2C_Type *base)`

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_master_flags`

Parameters

- `base` – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)
```

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

`_lpi2c_master_flags`.

Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_MasterGetStatusFlags()`.

```
static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Enables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Disables the LPI2C master interrupt requests.

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)
```

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- `base` – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)
 Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)
 Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)
 Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

```
void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t
                             baudRate_Hz)
```

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- `base` – The LPI2C peripheral base address.
- `sourceClock_Hz` – LPI2C functional clock frequency in Hertz.
- `baudRate_Hz` – Requested bus frequency in Hertz.

```
static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `true` – Bus is busy.
- `false` – Bus is idle.

```
status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, lpi2c_direction_t dir)
```

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

```
static inline status_t LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address,
                                                  lpi2c_direction_t dir)
```

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The LPI2C peripheral base address.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kLPI2C_Read` or `kLPI2C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

Parameters

- `base` – The LPI2C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was sent successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or over run.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- `base` – The LPI2C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.

- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterStop(LPI2C_Type *base)`

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The LPI2C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, lpi2c_master_transfer_t *transfer)`

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The LPI2C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_LPI2C_Busy` – Another master is currently utilizing the bus.
- `kStatus_LPI2C_Nak` – The slave device sent a NAK in response to a byte.
- `kStatus_LPI2C_FifoError` – FIFO under run or overrun.
- `kStatus_LPI2C_ArbitrationLost` – Arbitration lost error.
- `kStatus_LPI2C_PinLowTimeout` – SCL or SDA were held low longer than the timeout.

`void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, lpi2c_master_handle_t *handle, lpi2c_master_transfer_callback_t callback, void *userData)`

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `LPI2C_MasterTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *size_t* *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag

Bus busy flag

enumerator kLPI2C_MasterClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags

Errors to check for.

enum _lpi2c_direction

Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write

Master transmit.

enumerator kLPI2C_Read

Master receive.

enum _lpi2c_master_pin_config

LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain

LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly

LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull

LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull

LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave

LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave

LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave

LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput

LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source

LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin

Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger

Select the input trigger as the host request input

enum _lpi2c_host_request_polarity

LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow

Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh

Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode

LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled

LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1

LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1

LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1

LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1

LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1

LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1

LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Values:

enumerator kLPI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferNoStopFlag

Don't send a stop condition.

typedef enum _lpi2c_direction lpi2c_direction_t

Direction of master and slave transfers.


```
typedef enum _lpi2c_master_pin_config lpi2c_master_pin_config_t
```

LPI2C pin configuration.

```
typedef enum _lpi2c_host_request_source lpi2c_host_request_source_t
```

LPI2C master host request selection.

```
typedef enum _lpi2c_host_request_polarity lpi2c_host_request_polarity_t
```

LPI2C master host request pin polarity configuration.

```
typedef struct _lpi2c_master_config lpi2c_master_config_t
```

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _lpi2c_data_match_config_mode lpi2c_data_match_config_mode_t
```

LPI2C master data match configuration modes.

```
typedef struct _lpi2c_match_config lpi2c_data_match_config_t
```

LPI2C master data match configuration structure.

```
typedef struct _lpi2c_master_transfer lpi2c_master_transfer_t
```

LPI2C master descriptor of the transfer.

```
typedef struct _lpi2c_master_handle lpi2c_master_handle_t
```

LPI2C master handle of the transfer.

```
typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t  
*handle, status_t completionStatus, void *userData)
```

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base

The LPI2C peripheral base address.

Param handle

Pointer to the LPI2C master driver handle.

Param completionStatus

Either kStatus_Success or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)
```

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

```
struct _lpi2c_master_config
```

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns

Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest

Host request options.

struct *_lpi2c_match_config*

#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode

Data match configuration setting.

bool rxDataMatchOnly

When set to true, received data is ignored until a successful match.

uint32_t match0

Match value 0.

uint32_t match1

Match value 1.

struct *_lpi2c_master_transfer*

#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration `_lpi2c_master_transfer_flags` for available options. Set to 0 or `kLPI2C_TransferDefaultFlag` for normal transfers.

uint16_t slaveAddress

The 7-bit slave address.

lpi2c_direction_t direction

Either `kLPI2C_Read` or `kLPI2C_Write`.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

struct `_lpi2c_master_handle`

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint16_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: `Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]`

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

struct hostRequest

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.22 LPI2C Master DMA Driver

```
void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   edma_handle_t *rxDmaHandle, edma_handle_t
                                   *txDmaHandle, lpi2c_master_edma_transfer_callback_t
                                   callback, void *userData)
```

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

```
status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle,
                                   lpi2c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                           *handle, size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.
- *count* – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- *kStatus_Success* –
- *kStatus_NoTransferInProgress* – There is not a DMA transaction currently in progress.

```
status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, lpi2c_master_edma_handle_t
                                         *handle)
```

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to the LPI2C master driver handle.

Return values

- *kStatus_Success* – A transaction was successfully aborted.
- *kStatus_LPI2C_Idle* – There is not a DMA transaction currently in progress.

```
typedef struct lpi2c_master_edma_handle lpi2c_master_edma_handle_t
    LPI2C master EDMA handle of the transfer.
```

```
typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)
```

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to `LPI2C_MasterCreateEDMAHandle()`.

Param base

The LPI2C peripheral base address.

Param handle

Handle associated with the completed transfer.

Param completionStatus

Either *kStatus_Success* or an error code describing how the transfer completed.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct _lpi2c_master_edma_handle
    #include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.
```

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base
LPI2C base pointer.

bool isBusy
Transfer state machine current state.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[20]
LPI2C command sequence. When all 10 command words are used:
Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] +
receive&Size[4 words]

***lpi2c_master_transfer_t* transfer**
Copy of the current transfer info.

***lpi2c_master_edma_transfer_callback_t* completionCallback**
Callback function pointer.

void *userData
Application data passed to callback.

***edma_handle_t* *rx**
Handle for receive DMA channel.

***edma_handle_t* *tx**
Handle for transmit DMA channel.

***edma_tcd_t* tcds[3]**
Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.23 LPI2C Slave Driver

```
void LPI2C_SlaveGetDefaultConfig(lpi2c_slave_config_t *slaveConfig)
```

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave      = true;
slaveConfig->address0         = 0U;
slaveConfig->address1         = 0U;
slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable = true;
slaveConfig->filterEnable     = true;
slaveConfig->enableGeneralCall = false;
slaveConfig->sclStall.enableAck = false;
slaveConfig->sclStall.enableTx  = true;
slaveConfig->sclStall.enableRx  = true;
```

(continues on next page)

(continued from previous page)

```

slaveConfig->sclStall.enableAddress  = true;
slaveConfig->ignoreAck              = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns = 0;
slaveConfig->sclGlitchFilterWidth_ns = 0;
slaveConfig->dataValidDelay_ns      = 0;
slaveConfig->clockHoldTime_ns       = 0;

```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with `LPI2C_SlaveInit()`. Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure that is set to default values. Refer to `lpi2c_slave_config_t`.

```
void LPI2C_SlaveInit(LPI2C_Type *base, const lpi2c_slave_config_t *slaveConfig, uint32_t
                    sourceClock_Hz)
```

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The LPI2C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `LPI2C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

```
void LPI2C_SlaveDeinit(LPI2C_Type *base)
```

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveReset(LPI2C_Type *base)
```

Performs a software reset of the LPI2C slave peripheral.

Parameters

- `base` – The LPI2C peripheral base address.

```
static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)
```

Enables or disables the LPI2C module as slave.

Parameters

- `base` – The LPI2C peripheral base address.
- `enable` – Pass true to enable or false to disable the specified LPI2C as slave.

```
static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)
```

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_lpi2c_slave_flags`

Parameters

- `base` – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)
```

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- `kLPI2C_SlaveRepeatedStartDetectFlag`
- `kLPI2C_SlaveStopDetectFlag`
- `kLPI2C_SlaveBitErrFlag`
- `kLPI2C_SlaveFifoErrFlag`

Attempts to clear other flags has no effect.

See also:

`_lpi2c_slave_flags`.

Parameters

- `base` – The LPI2C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_lpi2c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `LPI2C_SlaveGetStatusFlags()`.

```
static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Enables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)
```

Disables the LPI2C slave interrupt requests.

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Parameters

- `base` – The LPI2C peripheral base address.
- `interruptMask` – Bit mask of interrupts to disable. See `_lpi2c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.


```
static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)
```

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of `_lpi2c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)
```

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

```
static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)
```

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

```
static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)
```

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the `kLPI2C_SlaveTransmitAckFlag` is asserted. This only happens if you enable the `sclStall.enableAck` field of the `lpi2c_slave_config_t` configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.
- ackOrNack – Pass true for an ACK or false for a NAK.

```
static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)
```

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL, false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns

Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns

Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, lpi2c_slave_handle_t *handle, lpi2c_slave_transfer_callback_t callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

```
status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, lpi2c_slave_handle_t *handle,
                                         uint32_t eventMask)
```

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kLPI2C_SlaveTransmitEvent* and *kLPI2C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to *lpi2c_slave_handle_t* structure which stores the transfer state.
- *eventMask* – Bit mask formed by OR'ing together *lpi2c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kLPI2C_SlaveAllEvents* to enable all events.

Return values

- *kStatus__Success* – Slave transfers were successfully started.
- *kStatus__LPI2C_Busy* – Slave transfers have already been started on this handle.

```
status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, lpi2c_slave_handle_t *handle, size_t
                                     *count)
```

Gets the slave transfer status during a non-blocking transfer.

Parameters

- *base* – The LPI2C peripheral base address.
- *handle* – Pointer to *i2c_slave_handle_t* structure.
- *count* – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- *kStatus__Success* –
- *kStatus__NoTransferInProgress* –

```
void LPI2C_SlaveTransferAbort(LPI2C_Type *base, lpi2c_slave_handle_t *handle)
```

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- *base* – The LPI2C peripheral base address.

- `handle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, lpi2c_slave_handle_t *handle)`

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The LPI2C peripheral base address.
- `handle` – Pointer to `lpi2c_slave_handle_t` structure which stores the transfer state.

`enum _lpi2c_slave_flags`

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- `kLPI2C_SlaveRepeatedStartDetectFlag`
- `kLPI2C_SlaveStopDetectFlag`
- `kLPI2C_SlaveBitErrFlag`
- `kLPI2C_SlaveFifoErrFlag`

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator `kLPI2C_SlaveTxReadyFlag`

Transmit data flag

enumerator `kLPI2C_SlaveRxReadyFlag`

Receive data flag

enumerator `kLPI2C_SlaveAddressValidFlag`

Address valid flag

enumerator `kLPI2C_SlaveTransmitAckFlag`

Transmit ACK flag

enumerator `kLPI2C_SlaveRepeatedStartDetectFlag`

Repeated start detect flag

enumerator `kLPI2C_SlaveStopDetectFlag`

Stop detect flag

enumerator `kLPI2C_SlaveBitErrFlag`

Bit error flag

enumerator `kLPI2C_SlaveFifoErrFlag`

FIFO error flag

enumerator `kLPI2C_SlaveAddressMatch0Flag`

Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag

Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag

General call flag

enumerator kLPI2C_SlaveBusyFlag

Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag

Bus busy flag

enumerator kLPI2C_SlaveClearFlags

All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags

Errors to check for.

enum _lpi2c_slave_address_match

LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0

Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

LPI2C slave handle structure.

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base

Base address for the LPI2C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

struct *_lpi2c_slave_config*

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

`bool enableSlave`
Enable slave mode.

`uint8_t address0`
Slave's 7-bit address.

`uint8_t address1`
Alternate slave 7-bit address.

`lpi2c_slave_address_match_t addressMatchMode`
Address matching options.

`bool filterDozeEnable`
Enable digital glitch filter in doze mode.

`bool filterEnable`
Enable digital glitch filter.

`bool enableGeneralCall`
Enable general call address matching.

`struct _lpi2c_slave_config sclStall`
SCL stall enable options.

`bool ignoreAck`
Continue transfers after a NACK is detected.

`bool enableReceivedAddressRead`
Enable reading the address received address as the first byte of data.

`uint32_t sdaGlitchFilterWidth_ns`
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

`uint32_t sclGlitchFilterWidth_ns`
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

`uint32_t dataValidDelay_ns`
Width in nanoseconds of the data valid delay.

`uint32_t clockHoldTime_ns`
Width in nanoseconds of the clock hold time.

`struct _lpi2c_slave_transfer`
#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

`lpi2c_slave_transfer_event_t event`
Reason the callback is being invoked.

`uint8_t receivedAddress`
Matching address send by master.

`uint8_t *data`
Transfer buffer

`size_t dataSize`
Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct *_lpi2c_slave_handle*

#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer

LPI2C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck

Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

2.24 LPSPI: Low Power Serial Peripheral Interface

2.25 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 $\text{delayTime} = \text{LPSPI\_clockSource} / (\text{PRESCALE} * \text{Delay\_scaler})$.

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nano-seconds.
- whichDelay – The desired delay to configuration, which must be of type *lpspi_delay_type_t*.
- srcClock_Hz – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

```
static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)
```

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result

in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- `base` – LPSPI peripheral address.
- `data` – The data word to be sent.

```
static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)
```

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- `base` – LPSPI peripheral address.

Returns

The data read from the data buffer.

```
void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)
```

Set up the dummy data.

Parameters

- `base` – LPSPI peripheral address.
- `dummyData` – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

```
void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, lpspi_master_handle_t *handle,  
                                     lpspi_master_transfer_callback_t callback, void  
                                     *userData)
```

Initializes the LPSPI master handle.

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

Parameters

- `base` – LPSPI peripheral address.
- `handle` – LPSPI handle pointer to `lpspi_master_handle_t`.
- `callback` – DSPI callback.
- `userData` – callback function parameter.

```
status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, lpspi_transfer_t *transfer)
```

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

- `base` – LPSPI peripheral address.
- `transfer` – pointer to `lpspi_transfer_t` structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.
- transfer – pointer to *lpspi_transfer_t* structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *size_t* *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_master_handle_t* structure which stores the transfer state.

```
void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, lpspi_slave_handle_t *handle,  
                                     lpspi_slave_transfer_callback_t callback, void *userData)
```

Initializes the LPSPI slave handle.

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

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to *lpspi_slave_handle_t*.
- callback – DSPI callback.
- userData – callback function parameter.

```
status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, lpspi_slave_handle_t *handle,  
                                         lpspi_transfer_t *transfer)
```

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.
- transfer – pointer to *lpspi_transfer_t* structure.

Returns

status of *status_t*.

```
status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, lpspi_slave_handle_t *handle, size_t  
                                     *count)
```

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of *status_t*.

```
void LPSPI_SlaveTransferAbort(LPSPI_Type *base, lpspi_slave_handle_t *handle)
```

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to *lpspi_slave_handle_t* structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns

true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum *_lpspi_flags*

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag

Data Match flag

enumerator kLPSPI_ModuleBusyFlag

Module Busy flag

enumerator kLPSPI_AllStatusFlag

Used for clearing all w1c status flags

enum _lpspi_interrupt_enable

LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable

Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable

Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable

Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable

Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable

Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable

Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable

Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable

Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable

All above interrupts enable.

enum _lpspi_dma_enable

LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0
PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity
LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum `_lpspi_clock_polarity`

LPSPI clock polarity configuration.

Values:

enumerator `kLPSPi_ClockPolarityActiveHigh`

CPOL=0. Active-high LPSPi clock (idles low)

enumerator `kLPSPi_ClockPolarityActiveLow`

CPOL=1. Active-low LPSPi clock (idles high)

enum `_lpspi_clock_phase`

LPSPi clock phase configuration.

Values:

enumerator `kLPSPi_ClockPhaseFirstEdge`

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator `kLPSPi_ClockPhaseSecondEdge`

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum `_lpspi_shift_direction`

LPSPi data shifter direction options.

Values:

enumerator `kLPSPi_MsbFirst`

Data transfers start with most significant bit.

enumerator `kLPSPi_LsbFirst`

Data transfers start with least significant bit.

enum `_lpspi_host_request_select`

LPSPi Host Request select configuration.

Values:

enumerator `kLPSPi_HostReqExtPin`

Host Request is an ext pin.

enumerator `kLPSPi_HostReqInternalTrigger`

Host Request is an internal trigger.

enum `_lpspi_match_config`

LPSPi Match configuration options.

Values:

enumerator `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 kLPSI_1stWordAndM1EqualsM0andM1
LPSPi Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1
LPSPi Match Enabled.

enum _lpspi_pin_config
LPSPi pin (SDO and SDI) configuration.

Values:

enumerator kLPSPI_SdiInSdoOut
LPSPi SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut
LPSPi SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut
LPSPi SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut
LPSPi SDO input, SDI output.

enum _lpspi_data_out_config
LPSPi data output configuration.

Values:

enumerator kLpspiDataOutRetained
Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate
Data out is tristated when chip select is de-asserted

enum _lpspi_transfer_width
LPSPi transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPi delay type selection.

Values:

enumerator kLPSPI_PcsToSck
PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
Delay between transfers.

enum _lpspi_transfer_config_flag_for_master

Use this enumeration for LPSPi master transfer configFlags.

Values:

enumerator kLPSPi_MasterPcs0

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS0 signal

enumerator kLPSPi_MasterPcs1

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS1 signal

enumerator kLPSPi_MasterPcs2

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS2 signal

enumerator kLPSPi_MasterPcs3

LPSPi master PCS shift macro , internal used. LPSPi master transfer use PCS3 signal

enumerator kLPSPi_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPi_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi_MasterByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPi slave transfer configFlags.

Values:

enumerator kLPSPi_SlavePcs0

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS0 signal

enumerator kLPSPi_SlavePcs1

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS1 signal

enumerator kLPSPi_SlavePcs2

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS2 signal

enumerator kLPSPi_SlavePcs3

LPSPi slave PCS shift macro , internal used. LPSPi slave transfer use PCS3 signal

enumerator kLPSPi_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

- i. If you set bitPerFrame = 8 , no matter the kLPSPi_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
- ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.
- iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPi_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPi_SlaveByteSwap flag.

enum `_lpspi_transfer_state`

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator `kLPSPI_Idle`

Nothing in the transmitter/receiver.

enumerator `kLPSPI_Busy`

Transfer queue is not finished.

enumerator `kLPSPI_Error`

Transfer error.

typedef enum `_lpspi_master_slave_mode` `lpspi_master_slave_mode_t`

LPSPI master or slave mode configuration.

typedef enum `_lpspi_which_pcs_config` `lpspi_which_pcs_t`

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum `_lpspi_pcs_polarity_config` `lpspi_pcs_polarity_config_t`

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum `_lpspi_clock_polarity` `lpspi_clock_polarity_t`

LPSPI clock polarity configuration.

typedef enum `_lpspi_clock_phase` `lpspi_clock_phase_t`

LPSPI clock phase configuration.

typedef enum `_lpspi_shift_direction` `lpspi_shift_direction_t`

LPSPI data shifter direction options.

typedef enum `_lpspi_host_request_select` `lpspi_host_request_select_t`

LPSPI Host Request select configuration.

typedef enum `_lpspi_match_config` `lpspi_match_config_t`

LPSPI Match configuration options.

typedef enum `_lpspi_pin_config` `lpspi_pin_config_t`

LPSPI pin (SDO and SDI) configuration.

typedef enum `_lpspi_data_out_config` `lpspi_data_out_config_t`

LPSPI data output configuration.

typedef enum `_lpspi_transfer_width` `lpspi_transfer_width_t`

LPSPI transfer width configuration.

typedef enum `_lpspi_delay_type` `lpspi_delay_type_t`

LPSPI delay type selection.

typedef struct `_lpspi_master_config` `lpspi_master_config_t`

LPSPI master configuration structure.

typedef struct `_lpspi_slave_config` `lpspi_slave_config_t`

LPSPI slave configuration structure.

typedef struct `_lpspi_master_handle` `lpspi_master_handle_t`

Forward declaration of the `_lpspi_master_handle` typedefs.

typedef struct `_lpspi_slave_handle` `lpspi_slave_handle_t`

Forward declaration of the `_lpspi_slave_handle` typedefs.

```
typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)
```

Master completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI master.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle, status_t status, void *userData)
```

Slave completion callback function pointer type.

Param base

LPSPI peripheral address.

Param handle

Pointer to the handle for the LPSPI slave.

Param status

Success or error code describing whether the transfer is completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
typedef struct _lpspi_transfer lpspi_transfer_t
```

LPSPI master/slave transfer structure.

```
volatile uint8_t g_lpspiDummyData[]
```

Global variable for dummy data value setting.

```
LPSPI_DUMMY_DATA
```

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

```
SPI_RETRY_TIMES
```

Retry times for waiting flag.

```
LPSPI_MASTER_PCS_SHIFT
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_MASTER_PCS_MASK
```

LPSPI master PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_SHIFT
```

LPSPI slave PCS shift macro , internal used.

```
LPSPI_SLAVE_PCS_MASK
```

LPSPI slave PCS shift macro , internal used.

```
struct _lpspi_master_config
```

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members`uint32_t baudRate`

Baud Rate for LPSPI.

`uint32_t bitsPerFrame`

Bits per frame, minimum 8, maximum 4096.

`lpspi_clock_polarity_t cpol`

Clock polarity.

`lpspi_clock_phase_t cpha`

Clock phase.

`lpspi_shift_direction_t direction`

MSB or LSB data shift direction.

`uint32_t pcsToSckDelayInNanoSec`

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`uint32_t lastSckToPcsDelayInNanoSec`

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`uint32_t betweenTransferDelayInNanoSec`

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

`lpspi_which_pcs_t whichPcs`

Desired Peripheral Chip Select (PCS).

`lpspi_pcs_polarity_config_t pcsActiveHighOrLow`

Desired PCS active high or low

`lpspi_pin_config_t pinCfg`

Configures which pins are used for input and output data during single bit transfers.

`lpspi_data_out_config_t dataOutConfig`

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

`bool enableInputDelay`

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

`struct __lpspi_slave_config`

#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members`uint32_t bitsPerFrame`

Bits per frame, minimum 8, maximum 4096.

`lpspi_clock_polarity_t cpol`

Clock polarity.

`lpspi_clock_phase_t cpha`

Clock phase.

`lpspi_shift_direction_t direction`

MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct *_lpspi_transfer*

#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize

Transfer bytes.

uint32_t configFlags

Transfer transfer configuration flags. Set from *_lpspi_transfer_config_flag_for_master* if the transfer is used for master or *_lpspi_transfer_config_flag_for_slave* enumeration if the transfer is used for slave.

struct *_lpspi_master_handle*

#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool writeTcrInIsr

A flag that whether should write TCR in ISR.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile bool isTxMask

A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame

Number of bytes in each frame

volatile uint16_t frameSize

Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

```
volatile uint8_t bytesEachWrite
    Bytes for each write TDR.
volatile uint8_t bytesEachRead
    Bytes for each read RDR.
const uint8_t *volatile txData
    Send buffer.
uint8_t *volatile rxData
    Receive buffer.
volatile size_t txRemainingByteCount
    Number of bytes remaining to send.
volatile size_t rxRemainingByteCount
    Number of bytes remaining to receive.
volatile uint32_t writeRegRemainingTimes
    Write TDR register remaining times.
volatile uint32_t readRegRemainingTimes
    Read RDR register remaining times.
uint32_t totalByteCount
    Number of transfer bytes
uint32_t txBuffIfNull
    Used if the txData is NULL.
volatile uint8_t state
    LPSPI transfer state , _lpspi_transfer_state.
lpspi_master_transfer_callback_t callback
    Completion callback.
void *userData
    Callback user data.
struct _lpspi_slave_handle
    #include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.
```

Public Members

```
volatile bool isByteSwap
    A flag that whether should byte swap.
volatile uint8_t fifoSize
    FIFO dataSize.
volatile uint8_t rxWatermark
    Rx watermark.
volatile uint8_t bytesEachWrite
    Bytes for each write TDR.
volatile uint8_t bytesEachRead
    Bytes for each read RDR.
const uint8_t *volatile txData
    Send buffer.
```

`uint8_t *volatile rxData`
Receive buffer.

`volatile size_t txRemainingByteCount`
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`volatile uint8_t state`
LPSPI transfer state , `_lpspi_transfer_state`.

`volatile uint32_t errorCount`
Error count for slave transfer.

`lpspi_slave_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

2.26 LPSPI eDMA Driver

`FSL_LPSPi_EDMA_DRIVER_VERSION`
LPSPI EDMA driver version.

`DMA_MAX_TRANSFER_COUNT`
DMA max transfer size.

`typedef struct _lpspi_master_edma_handle lpspi_master_edma_handle_t`
Forward declaration of the `_lpspi_master_edma_handle` typedefs.

`typedef struct _lpspi_slave_edma_handle lpspi_slave_edma_handle_t`
Forward declaration of the `_lpspi_slave_edma_handle` typedefs.

`typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base,
lpspi_master_edma_handle_t *handle, status_t status, void *userData)`
Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

```
typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPi_Type *base,
lpspi_slave_edma_handle_t *handle, status_t status, void *userData)
```

Completion callback function pointer type.

Param base

LPSPi peripheral base address.

Param handle

Pointer to the handle for the LPSPi slave.

Param status

Success or error code describing whether the transfer completed.

Param userData

Arbitrary pointer-dataSized value passed from the application.

```
void LPSPi_MasterTransferCreateHandleEDMA(LPSPi_Type *base, lpspi_master_edma_handle_t
*handle, lpspi_master_edma_transfer_callback_t
callback, void *userData, edma_handle_t
*edmaRxRegToRxDataHandle, edma_handle_t
*edmaTxDataToTxRegHandle)
```

Initializes the LPSPi master eDMA handle.

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

Note that the LPSPi eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPi peripheral base address.
- handle – LPSPi handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPi callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

```
status_t LPSPi_MasterTransferEDMA(LPSPi_Type *base, lpspi_master_edma_handle_t *handle,
lpspi_transfer_t *transfer)
```

LPSPi master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPi peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`status_t` LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- configFlags – transfer configuration flags. `_lpspi_transfer_config_flag_for_master`.

Return values

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.

Returns

Indicates whether LPSPI master transfer was successful or not.

`status_t` LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure, config field is not used.

Return values

- `kStatus_Success` – Execution successfully.
- `kStatus_LPSPI_Busy` – The LPSPI device is busy.
- `kStatus_InvalidArgument` – The transfer structure is invalid.

Returns

Indicates whether LPSPI master transfer was successful or not.

`void` LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *size_t* *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_master_edma_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns

status of *status_t*.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, *void* *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

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

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for `edmaRxRegToRxDataHandle`.

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to `lpspi_slave_edma_handle_t`.
- callback – LPSPI callback.
- userData – callback function parameter.
- `edmaRxRegToRxDataHandle` – `edmaRxRegToRxDataHandle` pointer to `edma_handle_t`.
- `edmaTxDataToTxRegHandle` – `edmaTxDataToTxRegHandle` pointer to `edma_handle_t`.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size

should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- transfer – pointer to `lpspi_transfer_t` structure.

Returns

status of `status_t`.

`void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)`

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.

`status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, size_t *count)`

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to `lpspi_slave_edma_handle_t` structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns

status of `status_t`.

`struct _lpspi_master_edma_handle`

`#include <fsl_lpspi_edma.h>` LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

`volatile bool isPcsContinuous`

Is PCS continuous in transfer.

`volatile bool isByteSwap`

A flag that whether should byte swap.

`volatile uint8_t fifoSize`

FIFO dataSize.

`volatile uint8_t rxWatermark`

Rx watermark.

`volatile uint8_t bytesEachWrite`

Bytes for each write TDR.

`volatile uint8_t bytesEachRead`
Bytes for each read RDR.

`volatile uint8_t bytesLastRead`
Bytes for last read RDR.

`volatile bool isThereExtraRxBytes`
Is there extra RX byte.

`const uint8_t *volatile txData`
Send buffer.

`uint8_t *volatile rxData`
Receive buffer.

`volatile size_t txRemainingByteCount`
Number of bytes remaining to send.

`volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`edma_tcd_t *lastTimeTCD`
Pointer to the lastTime TCD

`bool isMultiDMATransmit`
Is there multi DMA transmit

`volatile uint8_t dmaTransmitTime`
DMA Transfer times.

`uint32_t lastTimeDataBytes`
DMA transmit last Time data Bytes

`uint32_t dataBytesEveryTime`
Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

`edma_transfer_config_t transferConfigRx`
Config of DMA rx channel.

`edma_transfer_config_t transferConfigTx`
Config of DMA tx channel.

`uint32_t txBuffIfNull`
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`
Used if there is not rxData for DMA purpose.

`uint32_t transmitCommand`
Used to write TCR for DMA purpose.

`volatile uint8_t state`
LPSPI transfer state , `_lpspi_transfer_state`.

```
uint8_t nbytes
    eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
    Completion callback.

void *userData
    Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
    edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
    edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
    SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
    #include <fsl_lpspi_edma.h> LPSPi slave eDMA transfer handle structure used for transac-
    tional API.
```

Public Members

```
volatile bool isByteSwap
    A flag that whether should byte swap.

volatile uint8_t fifoSize
    FIFO dataSize.

volatile uint8_t rxWatermark
    Rx watermark.

volatile uint8_t bytesEachWrite
    Bytes for each write TDR.

volatile uint8_t bytesEachRead
    Bytes for each read RDR.

volatile uint8_t bytesLastRead
    Bytes for last read RDR.

volatile bool isThereExtraRxBytes
    Is there extra RX byte.

uint8_t nbytes
    eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData
    Send buffer.

uint8_t *volatile rxData
    Receive buffer.

volatile size_t txRemainingByteCount
    Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
    Number of bytes remaining to receive.
```

`volatile uint32_t writeRegRemainingTimes`
Write TDR register remaining times.

`volatile uint32_t readRegRemainingTimes`
Read RDR register remaining times.

`uint32_t totalByteCount`
Number of transfer bytes

`uint32_t txBuffIfNull`
Used if there is not txData for DMA purpose.

`uint32_t rxBuffIfNull`
Used if there is not rxData for DMA purpose.

`volatile uint8_t state`
LPSPI transfer state.

`uint32_t errorCount`
Error count for slave transfer.

`lpspi_slave_edma_transfer_callback_t callback`
Completion callback.

`void *userData`
Callback user data.

`edma_handle_t *edmaRxRegToRxDataHandle`
edma_handle_t handle point used for RxReg to RxData buff

`edma_handle_t *edmaTxDataToTxRegHandle`
edma_handle_t handle point used for TxData to TxReg

`edma_tcd_t lpspiSoftwareTCD[2]`
SoftwareTCD, internal used

2.27 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

2.28 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.29 LPUART Driver

`static inline void LPUART_SoftwareReset(LPUART_Type *base)`

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- `base` – LPUART peripheral base address.

`status_t LPUART_Init(LPUART_Type *base, const lpuart_config_t *config, uint32_t srcClock_Hz)`

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the `LPUART_GetDefaultConfig()` function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- `base` – LPUART peripheral base address.
- `config` – Pointer to a user-defined configuration structure.
- `srcClock_Hz` – LPUART clock source frequency in HZ.

Return values

- `kStatus_LPUART_BaudrateNotSupport` – Baudrate is not support in current clock source.
- `kStatus_Success` – LPUART initialize succeed

`status_t LPUART_Deinit(LPUART_Type *base)`

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- `base` – LPUART peripheral base address.

Return values

- `kStatus_Success` – Deinit is success.
- `kStatus_LPUART_Timeout` – Timeout during deinit.

```
void LPUART_GetDefaultConfig(lpuart_config_t *config)
```

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: `lpuartConfig->baudRate_Bps = 115200U`; `lpuartConfig->parityMode = kLPUART_ParityDisabled`; `lpuartConfig->dataBitsCount = kLPUART_EightDataBits`; `lpuartConfig->isMsb = false`; `lpuartConfig->stopBitCount = kLPUART_OneStopBit`; `lpuartConfig->txFifoWatermark = 0`; `lpuartConfig->rxFifoWatermark = 1`; `lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit`; `lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1`; `lpuartConfig->enableTx = false`; `lpuartConfig->enableRx = false`;

Parameters

- `config` – Pointer to a configuration structure.

```
status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t  
srcClock_Hz)
```

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the `LPUART_Init`.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- `base` – LPUART peripheral base address.
- `baudRate_Bps` – LPUART baudrate to be set.
- `srcClock_Hz` – LPUART clock source frequency in HZ.

Return values

- `kStatus_LPUART_BaudrateNotSupport` – Baudrate is not supported in the current clock source.
- `kStatus_Success` – Set baudrate succeeded.

```
void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)
```

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- `base` – LPUART peripheral base address.
- `enable` – true to enable, false to disable.

```
static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t  
address2)
```

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer; otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

```
static inline void LPUART__EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)
```

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

```
static inline void LPUART__SetRxFifoWatermark(LPUART_Type *base, uint8_t water)
```

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

```
static inline void LPUART__SetTxFifoWatermark(LPUART_Type *base, uint8_t water)
```

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

```
static inline void LPUART__TransferEnable16Bit(lpuart_handle_t *handle, bool enable)
```

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in `lpuart_handle_t`.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

```
uint32_t LPUART__GetStatusFlags(LPUART_Type *base)
```

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators `_lpuart_flags`. To check for a specific status, compare the return value with enumerators in the `_lpuart_flags`. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART__GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the `_lpuart_flags`.

`status_t` LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: `kLPUART_TxDataRegEmptyFlag`, `kLPUART_TransmissionCompleteFlag`, `kLPUART_RxDataRegFullFlag`, `kLPUART_RxActiveFlag`, `kLPUART_NoiseErrorFlag`, `kLPUART_ParityErrorFlag`, `kLPUART_TxFifoEmptyFlag`, `kLPUART_RxFifoEmptyFlag`. Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the `_lpuart_status_flag_t` to do the OR operation and get the mask.

Return values

- `kStatus_LPUART_FlagCannotClearManually` – The flag can't be cleared by this function but it is cleared automatically by hardware.
- `kStatus_Success` – Status in the mask are cleared.

Returns

0 succeed, others failed.

`void` LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the `_lpuart_interrupt_enable`. This example shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of `_lpuart_interrupt_enable`.

`void` LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpuart_interrupt_enable`. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1, kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↳ RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of `_lpuart_interrupt_enable`.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators `_lpuart_interrupt_enable`. To check a specific interrupt enable status, compare the return value with enumerators in `_lpuart_interrupt_enable`. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in `_lpuart_interrupt_enable`.

static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

`void LPUART_SendAddress(LPUART_Type *base, uint8_t address)`

Transmit an address frame in 9-bit data mode.

Parameters

- `base` – LPUART peripheral base address.
- `address` – LPUART slave address.

`status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)`

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the data to write.
- `length` – Size of the data to write.

Return values

- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully wrote all data.

`status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)`

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the data to write.
- `length` – Size of the data to write.

Return values

- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully wrote all data.

`status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)`

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the buffer to store the received data.
- `length` – Size of the buffer.

Return values

- `kStatus_LPUART_RxHardwareOverrun` – Receiver overrun happened while receiving data.
- `kStatus_LPUART_NoiseError` – Noise error happened while receiving data.

- `kStatus_LPUART_FramingError` – Framing error happened while receiving data.
- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)`

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- `base` – LPUART peripheral base address.
- `data` – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- `length` – Size of the buffer.

Return values

- `kStatus_LPUART_RxHardwareOverrun` – Receiver overrun happened while receiving data.
- `kStatus_LPUART_NoiseError` – Noise error happened while receiving data.
- `kStatus_LPUART_FramingError` – Framing error happened while receiving data.
- `kStatus_LPUART_ParityError` – Parity error happened while receiving data.
- `kStatus_LPUART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`void LPUART_TransferCreateHandle(LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_callback_t callback, void *userData)`

Initializes the LPUART handle.

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

The LPUART driver supports the “background” receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the `LPUART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

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

```
status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, lpuart_handle_t *handle,  
                                         lpuart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

```
void LPUART_TransferStartRingBuffer(LPUART_Type *base, lpuart_handle_t *handle, uint8_t  
                                     *ringBuffer, size_t ringBufferSize)
```

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

```
void LPUART_TransferStopRingBuffer(LPUART_Type *base, lpuart_handle_t *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.

- *handle* – LPUART handle pointer.

`size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, lpuart_handle_t *handle)`

Get the length of received data in RX ring buffer.

Parameters

- *base* – LPUART peripheral base address.
- *handle* – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

`void LPUART_TransferAbortSend(LPUART_Type *base, lpuart_handle_t *handle)`

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the `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(LPUART_Type *base, void *irqHandle)`

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

`void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)`

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_DriverIRQHandler(uint32_t instance)

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- instance – LPUART instance.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually

Some flag can't manually clear

enumerator kStatus_LPUART_Error

Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun

LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun

LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError

LPUART noise error.

enumerator kStatus_LPUART_FramingError

LPUART framing error.

enumerator kStatus_LPUART_ParityError

LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport

Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected

IDLE flag.

enumerator kStatus_LPUART_Timeout

LPUART times out.

enum _lpuart_parity_mode

LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled

Parity disabled

enumerator kLPUART_ParityEven

Parity enabled, type even, bit setting: PE | PT = 10

enumerator kLPUART_ParityOdd

Parity enabled, type odd, bit setting: PE | PT = 11

enum _lpuart_data_bits

LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits

Eight data bit

enumerator kLPUART_SevenDataBits

Seven data bit

enum _lpuart_stop_bit_count

LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit

One stop bit

enumerator kLPUART_TwoStopBit

Two stop bits

enum _lpuart_transmit_cts_source

LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin

CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult

CTS resource is the match result.

enum _lpuart_transmit_cts_config

LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart

CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle

CTS input is sampled when the transmitter is idle

enum _lpuart_transmit_rts_polarity

LPUART transmitter RTS polarity.

Values:

enumerator kLPUART_RtsPolarityLow

Transmitter RTS is active low.

enumerator kLPUART_RtsPolarityHigh

Transmitter RTS is active high.

enum _lpuart_idle_type_select

LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit

Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit

Start counting after a stop bit.

enum _lpuart_idle_config

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1

the number of idle characters.

enumerator kLPUART_IdleCharacter2

the number of idle characters.

enumerator kLPUART_IdleCharacter4

the number of idle characters.

enumerator kLPUART_IdleCharacter8

the number of idle characters.

enumerator kLPUART_IdleCharacter16

the number of idle characters.

enumerator kLPUART_IdleCharacter32

the number of idle characters.

enumerator kLPUART_IdleCharacter64

the number of idle characters.

enumerator kLPUART_IdleCharacter128

the number of idle characters.

enum _lpuart_interrupt_enable

LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable

LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable

Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable

Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable

Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags

LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection. bit 16

enumerator `kLPUART_LinBreakFlag`

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator `kLPUART_RxActiveEdgeFlag`

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator `kLPUART_RxActiveFlag`

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator `kLPUART_DataMatch1Flag`

The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator `kLPUART_DataMatch2Flag`

The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator `kLPUART_TxFifoEmptyFlag`

TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator `kLPUART_RxFifoEmptyFlag`

RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator `kLPUART_TxFifoOverflowFlag`

TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator `kLPUART_RxFifoUnderflowFlag`

RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator `kLPUART_AllClearFlags`

enumerator `kLPUART_AllFlags`

typedef enum *lpuart_parity_mode* `lpuart_parity_mode_t`

LPUART parity mode.

typedef enum *lpuart_data_bits* `lpuart_data_bits_t`

LPUART data bits count.

typedef enum *lpuart_stop_bit_count* `lpuart_stop_bit_count_t`

LPUART stop bit count.

typedef enum *lpuart_transmit_cts_source* `lpuart_transmit_cts_source_t`

LPUART transmit CTS source.

typedef enum *lpuart_transmit_cts_config* `lpuart_transmit_cts_config_t`

LPUART transmit CTS configure.

typedef enum *lpuart_transmit_rts_polarity* `lpuart_transmit_rts_polarity_t`

LPUART transmitter RTS polarity.

typedef enum *lpuart_idle_type_select* `lpuart_idle_type_select_t`

LPUART idle flag type defines when the receiver starts counting.

typedef enum *lpuart_idle_config* `lpuart_idle_config_t`

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *lpuart_config* `lpuart_config_t`

LPUART configuration structure.

typedef struct *lpuart_transfer* `lpuart_transfer_t`

LPUART transfer structure.

```
typedef struct lpuart_handle lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle,
status_t status, void *userData)
    LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
    Retry times for waiting flag.

struct _lpuart_config
    #include <fsl_lpuart.h> LPUART configuration structure.
```

Public Members

```
uint32_t baudRate_Bps
    LPUART baud rate

lpuart_parity_mode_t parityMode
    Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
    Data bits count, eight (default), seven

bool isMsb
    Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
    Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
    TX FIFO watermark

uint8_t rxFifoWatermark
    RX FIFO watermark

bool enableRxRTS
    RX RTS enable

bool enableTxRTS
    TX 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_transmit_rts_polarity_t txRtsPolarity
    TX RTS polarity
```

uint8_t rtsWatermark

RTS watermark

lpuart_idle_type_select_t rxIdleType

RX IDLE type.

lpuart_idle_config_t rxIdleConfig

RX IDLE configuration.

bool enableTx

Enable TX

bool enableRx

Enable RX

bool swapTxdRxd

Swap TXD and RXD pins

bool inverseTxd

Transmit Data Inversion - Setting true reverses the polarity of the transmitted data output

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 __unnamed21__`

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 __unnamed23__`

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 __unnamed25__`

Public Members

`uint8_t *volatile rxData`
Address of remaining data to receive.

`uint16_t *volatile rxData16`
Address of remaining data to receive.

`union __unnamed27__`

Public Members

uint8_t *rxRingBuffer
Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16
Start address of the receiver ring buffer.

2.30 LPUART eDMA Driver

```
void LPUART__TransferCreateHandleEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                                     lpuart_edma_transfer_callback_t callback, void  
                                     *userData, edma_handle_t *txEdmaHandle,  
                                     edma_handle_t *rxEdmaHandle)
```

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

```
status_t LPUART__SendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                          lpuart_transfer_t *xfer)
```

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

```
status_t LPUART__ReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle,  
                             lpuart_transfer_t *xfer)
```

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.
- xfer – LPUART eDMA transfer structure, see `lpuart_transfer_t`.

Return values

- `kStatus__Success` – if succeed, others fail.
- `kStatus__LPUART_RxBusy` – Previous transfer ongoing.
- `kStatus__InvalidArgument` – Invalid argument.

`void LPUART__TransferAbortSendEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`
Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`void LPUART__TransferAbortReceiveEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle)`
Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to `lpuart_edma_handle_t` structure.

`status_t LPUART__TransferGetSendCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- `kStatus__NoTransferInProgress` – No send in progress.
- `kStatus__InvalidArgument` – Parameter is invalid.
- `kStatus__Success` – Get successfully through the parameter count;

`status_t LPUART__TransferGetReceiveCountEDMA(LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)`

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- `kStatus_NoTransferInProgress` – No receive in progress.
- `kStatus_InvalidArgument` – Parameter is invalid.
- `kStatus_Success` – Get successfully through the parameter count;

`void LPUART__TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)`
LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- `base` – LPUART peripheral base address.
- `lpuartEdmaHandle` – LPUART handle pointer.

`FSL_LPUART_EDMA_DRIVER_VERSION`

LPUART EDMA driver version.

`typedef struct _lpuart_edma_handle lpuart_edma_handle_t`

`typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`

LPUART transfer callback function.

`struct _lpuart_edma_handle`

`#include <fsl_lpuart_edma.h>` LPUART eDMA handle.

Public Members

`lpuart_edma_transfer_callback_t` callback

Callback function.

`void *userData`

LPUART callback function parameter.

`size_t rxDataSizeAll`

Size of the data to receive.

`size_t txDataSizeAll`

Size of the data to send out.

`edma_handle_t *txEdmaHandle`

The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`

The eDMA RX channel used.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

`volatile uint8_t txState`

TX transfer state.

`volatile uint8_t rxState`

RX transfer state

2.31 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_SCGRReset

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.

```
CMC_BLR_LOCK_FIELD_WIDTH
```

```
CMC_BLR_LOCK_IDX_MASK(index)
```

```
CMC_BLR_LOCK_IDX_SHIFT(index)
```

```
CMC_BLR_LOCK_IDX(index, value)
```

```
void CMC_SetClockMode(CMC_Type *base, cmc_clock_mode_t mode)
```

Sets clock mode.

This function config 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 domain.

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 domain 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_Config 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 uint32_t CMC_GetBootRomStatus(CMC_Type *base, uint8_t index)
```

Gets the information written by the BootROM.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.

Returns

The status information written by the BootROM.

```
static inline void CMC_WriteBootRomStatusReg(CMC_Type *base, uint8_t index, uint32_t value)
```

Writes value to BootROM status register, in this way, BootROM status registers are used as general purpose register.

Note: Value in BootROM status registers are reset in cold reset.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.
- value – Value to write.

```
static inline void CMC_LockWriteOperationToBootRomStatusReg(CMC_Type *base, uint8_t  
                                                             index)
```

Lock write operation to BootROM status register and BootROM Lock register.

Note: If locked, BootROM status register cannot be written.

Note: Once locked, only cold reset can reset related register.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.

```
static inline bool CMC_CheckBootRomStatusRegWriteLocked(CMC_Type *base, uint8_t index)
```

Check if BootROM status register can be written.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.

Return values

- true – The selected BootRom status register is locked and cannot be written.
- false – The selected BootRom Status register is unlocked and cannot be written.

```
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)
Config the low power mode of the on-chip flash memory.

This function config 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.

2.32 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_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_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 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 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_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_CORELDIVoltageSetFail` – 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_CORELDOLowVoltageSetFail* – Fail to change Core LDO voltage level.

```
status_t SPC_SetLowPowerModeCoreLDOLowVoltageLevel(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_CORELDOVoltageWrong` – 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 configs 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.

- Configure `ACTIVE_CFG[DCDC_VDD_LVL]` to same level programmed in #1.

Note: To change DCDC level in Low-Power mode:

- Configure `LP_CFG[DCDC_VDD_LVL]` to desired level;
- 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 2.12.0.

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_CORELDIVoltageWrong`

Core LDO voltage is wrong.

enumerator `kStatus_SPC_CORELDIVoltageSetFail`

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_controlVbat
Enable/disable VBAT 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_controlTsi0

Enable/disable TSI0 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_Core_LDO_RetentionVoltage as instead.

enumerator kSPC_Core_LDO_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_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_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_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_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 HVDRestEnable

CORE/SYS/IO VDD High Voltage Detect reset enable.

bool LVDInterruptEnable

CORE/SYS/IO VDD Low Voltage Detect interrupt enable.

bool LVDRestEnable

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.

2.33 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_AllWakeupEnable

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 enabel 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 enabled.

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

- `kStatus_Success` – 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.

2.34 OPAMP: Operational Amplifier

`void OPAMP_Init(OPAMP_Type *base, const opamp_config_t *config)`

Initialize OPAMP instance.

Parameters

- `base` – OPAMP peripheral base address.
- `config` – The pointer to `opamp_config_t`.

`void OPAMP_Deinit(OPAMP_Type *base)`

De-initialize OPAMP instance.

Parameters

- `base` – OPAMP peripheral base address.

`void OPAMP_GetDefaultConfig(opamp_config_t *config)`

Get default configuration of OPAMP.

```
config->enable      = false;
config->mode        = kOPAMP_LowNoiseMode;
config->trimOption   = kOPAMP_TrimOptionDefault;
config->intRefVoltage = kOPAMP_IntRefVoltVddaDiv2;
config->enablePosADCSw = false;
config->posRefVoltage = kOPAMP_PosRefVoltVrefh3;
config->posGain      = kOPAMP_PosGainReserved;
config->negGain      = kOPAMP_NegGainBufferMode;
```

Parameters

- `config` – The pointer to `opamp_config_t`.

`static inline void OPAMP_DoPosGainConfig(OPAMP_Type *base, opamp_positive_gain_t option)`

Configure OPAMP positive port gain.

Parameters

- `base` – OPAMP peripheral base address.
- `option` – OPAMP positive port gain.

`static inline void OPAMP_DoNegGainConfig(OPAMP_Type *base, opamp_negative_gain_t option)`

Configure OPAMP negative port gain.

Parameters

- `base` – OPAMP peripheral base address.
- `option` – OPAMP negative port gain.

static inline void OPAMP_EnableRefBuffer(OPAMP_Type *base, bool enable)

Enable reference buffer.

Parameters

- base – OPAMP peripheral base address.
- enable – true to enable and false to disable.

static inline void OPAMP_EnableTriggerMode(OPAMP_Type *base, bool enable)

Enable OPAMP trigger mode.

Parameters

- base – OPAMP peripheral base address.
- enable – true to enable and false to disable.

FSL_OPAMP_DRIVER_VERSION

OPAMP driver version.

enum _opamp_mode

The enumeration of OPAMP mode, including low noise mode and high speed mode.

Values:

enumerator kOPAMP_LowNoiseMode

Set opamp mode as low noise mode.

enumerator kOPAMP_HighSpeedMode

Set opamp mode as high speed mode.

enum _opamp_bias_current_trim_option

The enumeration of bias current trim option.

Values:

enumerator kOPAMP_TrimOptionDefault

Default Bias current trim option.

enumerator kOPAMP_TrimOptionIncreaseCurrent

Trim option selected as increase current.

enumerator kOPAMP_TrimOptionDecreaseCurrent

Trim option selected as decrease current.

enumerator kOPAMP_TrimOptionFurtherDecreaseCurrent

Trim option selected as further decrease current.

enum _opamp_internal_ref_voltage

The enumeration of internal reference voltage.

Values:

enumerator kOPAMP_IntRefVoltVddaDiv2

Internal reference voltage selected as Vdda/2.

enumerator kOPAMP_IntRefVoltVdda3V

Internal reference voltage selected as Vdda_3V.

enumerator kOPAMP_IntRefVoltVssa3V

Internal reference voltage selected as Vssa_3V.

enumerator kOPAMP_IntRefVoltNotAllowed

Internal reference voltage not allowed.

enum _opamp_positive_ref_voltage

The enumeration of positive reference voltage—please refer to manual use—.

Values:

enumerator kOPAMP_PosRefVoltVrefh3

Positive part reference voltage select Vrefh3, connected from DAC output.

enumerator kOPAMP_PosRefVoltVrefh0

Positive part reference voltage select Vrefh0, connected from VDDA supply.

enumerator kOPAMP_PosRefVoltVrefh1

Positive part reference voltage select Vrefh1, connected from Voltage reference output.

enumerator kOPAMP_PosRefVoltVrefh4

Positive part reference voltage select 520mv or reserved.

enum _opamp_positive_gain

The enumeration of positive programmable gain (please refer to manual use).

Values:

enumerator kOPAMP_PosGainReserved

Positive Gain reserved.

enumerator kOPAMP_PosGainNonInvert1X

Positive non-inverting gain application 1X.

enumerator kOPAMP_PosGainNonInvert2X

Positive non-inverting gain application 2X.

enumerator kOPAMP_PosGainNonInvert4X

Positive non-inverting gain application 4X.

enumerator kOPAMP_PosGainNonInvert8X

Positive non-inverting gain application 8X.

enumerator kOPAMP_PosGainNonInvert16X

Positive non-inverting gain application 16X.

enumerator kOPAMP_PosGainNonInvert33X

Positive non-inverting gain application 33X.

enumerator kOPAMP_PosGainNonInvert64X

Positive non-inverting gain application 64X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer2X

Positive non-inverting gain application 2X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer3X

Positive non-inverting gain application 3X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer5X

Positive non-inverting gain application 5X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer9X

Positive non-inverting gain application 9X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer17X

Positive non-inverting gain application 17X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer34X

Positive non-inverting gain application 34X.

enumerator kOPAMP_PosGainNonInvertDisableBuffer65X
Positive non-inverting gain application 65X.

enum _opamp_negative_gain

The enumeration of negative programmable gain.

Values:

enumerator kOPAMP_NegGainBufferMode
Negative Buffer Mode.

enumerator kOPAMP_NegGainInvert1X
Negative inverting gain application -1X.

enumerator kOPAMP_NegGainInvert2X
Negative inverting gain application -2X.

enumerator kOPAMP_NegGainInvert4X
Negative inverting gain application -4X.

enumerator kOPAMP_NegGainInvert8X
Negative inverting gain application -8X.

enumerator kOPAMP_NegGainInvert16X
Negative inverting gain application -16X.

enumerator kOPAMP_NegGainInvert33X
Negative inverting gain application -33X.

enumerator kOPAMP_NegGainInvert64X
Negative inverting gain application -64X.

enum _opamp_positive_input_channel_selection

The enumeration of positive input channel selection.

Values:

enumerator kOPAMP_PosInputChannel0
When OPAMP not in trigger mode, select positive input 0 (INP0).

enumerator kOPAMP_PosInputChannel1
When OPAMP not in trigger mode, select positive input 1 (INP1).

typedef enum _opamp_mode opamp_mode_t

The enumeration of OPAMP mode, including low noise mode and high speed mode.

typedef enum _opamp_bias_current_trim_option opamp_bias_current_trim_option_t

The enumeration of bias current trim option.

typedef enum _opamp_internal_ref_voltage opamp_internal_ref_voltage_t

The enumeration of internal reference voltage.

typedef enum _opamp_positive_ref_voltage opamp_positive_ref_voltage_t

The enumeration of positive reference voltage□please refer to manual use□.

typedef enum _opamp_positive_gain opamp_positive_gain_t

The enumeration of positive programmable gain (please refer to manual use).

typedef enum _opamp_negative_gain opamp_negative_gain_t

The enumeration of negative programmable gain.

typedef enum _opamp_positive_input_channel_selection

opamp_positive_input_channel_selection_t

The enumeration of positive input channel selection.

```
typedef struct _opamp_config opamp_config_t
    OPAMP configuraion, including mode, internal reference voltage, positive gain, negative
    gain and so on.

struct _opamp_config
    #include <fsl_opamp.h> OPAMP configuraion, including mode, internal reference voltage,
    positive gain, negative gain and so on.
```

Public Members

```
bool enable
    Enable/disable OPAMP.

opamp_mode_t mode
    Opamp mode, available values are kOPAMP_LowNoiseMode and
    kOPAMP_HighSpeedMode.

opamp_bias_current_trim_option_t trimOption
    Bias current trim option, please refer to opamp_bias_current_trim_option_t.

opamp_internal_ref_voltage_t intRefVoltage
    Internal reference voltage, please refer to opamp_internal_ref_voltage_t.

opamp_positive_ref_voltage_t posRefVoltage
    Positive part reference voltage, please refer to opamp_positive_ref_voltage_t.

opamp_positive_gain_t posGain
    Positive part programmable gain, please refer to opamp_positive_gain_t.

opamp_negative_gain_t negGain
    Negative part programmable gain, please refer to opamp_negative_gain_t.

bool enableOutputSwitch
    OPAMP out to negative gain resistor ladder switch.

bool enablePosADCSw1
    Positive part reference voltage switch to ADC channel or not.
    • true Positive part reference voltage switch to ADC channel.
    • false Positive part reference voltage do not switch to ADC channel.

bool enablePosADCSw2
    Positive part reference voltage switch to ADC channel or not.
    • true Positive part reference voltage switch to ADC channel.
    • false Positive part reference voltage do not switch to ADC channel.

bool enableRefBuffer
    Reference buffer enable.

opamp_positive_input_channel_selection_t PosInputChannelSelection
    Positive Input Channel Selection

bool enableTriggerMode
    Trigger Mode Enable.
```

2.35 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,
```

(continues on next page)

(continued from previous page)

```

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_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

static inline void PORT_ClearAllHighEFTDetectors(PORT_Type *base)

Clear all high EFT detector.

Parameters

- base – PORT peripheral base pointer

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_invet_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.

```

2.36 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 2.9.4

enum *_pwm_submodule*

List of PWM submodules.

Values:

enumerator `kPWM_Module_0`

Submodule 0

enumerator `kPWM_Module_1`

Submodule 1

enumerator kPWM_Module_2
Submodule 2

enum _pwm_channels
List of PWM channels in each module.
Values:

enumerator kPWM_PwmB
enumerator kPWM_PwmA
enumerator kPWM_PwmX

enum _pwm_value_register
List of PWM value registers.
Values:

enumerator kPWM_ValueRegister_0
PWM Value0 register
enumerator kPWM_ValueRegister_1
PWM Value1 register
enumerator kPWM_ValueRegister_2
PWM Value2 register
enumerator kPWM_ValueRegister_3
PWM Value3 register
enumerator kPWM_ValueRegister_4
PWM Value4 register
enumerator kPWM_ValueRegister_5
PWM Value5 register

enum _pwm_value_register_mask
List of PWM value registers mask.
Values:

enumerator kPWM_ValueRegisterMask_0
PWM Value0 register mask
enumerator kPWM_ValueRegisterMask_1
PWM Value1 register mask
enumerator kPWM_ValueRegisterMask_2
PWM Value2 register mask
enumerator kPWM_ValueRegisterMask_3
PWM Value3 register mask
enumerator kPWM_ValueRegisterMask_4
PWM Value4 register mask
enumerator kPWM_ValueRegisterMask_5
PWM Value5 register mask

enum _pwm_clock_source
PWM clock source selection.
Values:

enumerator kPWM__BusClock

Device specific IPBus clock, refer reference manual for frequency

enumerator kPWM__ExternalClock

EXT_CLK is used as the clock

enumerator kPWM__Submodule0Clock

Clock of the submodule 0 (AUX_CLK) is used as the source clock

enum __pwm_clock_prescale

PWM prescaler factor selection for clock source.

Values:

enumerator kPWM__Prescale_Divide_1

PWM clock frequency = fclk/1

enumerator kPWM__Prescale_Divide_2

PWM clock frequency = fclk/2

enumerator kPWM__Prescale_Divide_4

PWM clock frequency = fclk/4

enumerator kPWM__Prescale_Divide_8

PWM clock frequency = fclk/8

enumerator kPWM__Prescale_Divide_16

PWM clock frequency = fclk/16

enumerator kPWM__Prescale_Divide_32

PWM clock frequency = fclk/32

enumerator kPWM__Prescale_Divide_64

PWM clock frequency = fclk/64

enumerator kPWM__Prescale_Divide_128

PWM clock frequency = fclk/128

enum __pwm_force_output_trigger

Options that can trigger a PWM FORCE_OUT.

Values:

enumerator kPWM__Force_Local

The local force signal, CTRL2[FORCE], from the submodule is used to force updates

enumerator kPWM__Force_Master

The master force signal from submodule 0 is used to force updates

enumerator kPWM__Force_LocalReload

The local reload signal from this submodule is used to force updates without regard to the state of LDOK

enumerator kPWM__Force_MasterReload

The master reload signal from submodule 0 is used to force updates if LDOK is set

enumerator kPWM__Force_LocalSync

The local sync signal from this submodule is used to force updates

enumerator kPWM__Force_MasterSync

The master sync signal from submodule0 is used to force updates

enumerator kPWM__Force_External

The external force signal, EXT_FORCE, from outside the PWM module causes updates

enumerator kPWM_Force_ExternalSync

The external sync signal, EXT_SYNC, from outside the PWM module causes updates

enum __pwm_output_state

PWM channel output status.

Values:

enumerator kPWM_HighState

The output state of PWM channel is high

enumerator kPWM_LowState

The output state of PWM channel is low

enumerator kPWM_NormalState

The output state of PWM channel is normal

enumerator kPWM_InvertState

The output state of PWM channel is invert

enumerator kPWM_MaskState

The output state of PWM channel is mask

enum __pwm_init_source

PWM counter initialization options.

Values:

enumerator kPWM_Initialize_LocalSync

Local sync causes initialization

enumerator kPWM_Initialize_MasterReload

Master reload from submodule 0 causes initialization

enumerator kPWM_Initialize_MasterSync

Master sync from submodule 0 causes initialization

enumerator kPWM_Initialize_ExtSync

EXT_SYNC causes initialization

enum __pwm_load_frequency

PWM load frequency selection.

Values:

enumerator kPWM_LoadEveryOpportunity

Every PWM opportunity

enumerator kPWM_LoadEvery2Opportunity

Every 2 PWM opportunities

enumerator kPWM_LoadEvery3Opportunity

Every 3 PWM opportunities

enumerator kPWM_LoadEvery4Opportunity

Every 4 PWM opportunities

enumerator kPWM_LoadEvery5Opportunity

Every 5 PWM opportunities

enumerator kPWM_LoadEvery6Opportunity

Every 6 PWM opportunities

enumerator kPWM_LoadEvery7Opportunity

Every 7 PWM opportunities

enumerator kPWM_LoadEvery8Opportunity

Every 8 PWM opportunities

enumerator kPWM_LoadEvery9Opportunity

Every 9 PWM opportunities

enumerator kPWM_LoadEvery10Opportunity

Every 10 PWM opportunities

enumerator kPWM_LoadEvery11Opportunity

Every 11 PWM opportunities

enumerator kPWM_LoadEvery12Opportunity

Every 12 PWM opportunities

enumerator kPWM_LoadEvery13Opportunity

Every 13 PWM opportunities

enumerator kPWM_LoadEvery14Opportunity

Every 14 PWM opportunities

enumerator kPWM_LoadEvery15Opportunity

Every 15 PWM opportunities

enumerator kPWM_LoadEvery16Opportunity

Every 16 PWM opportunities

enum __pwm_fault_input

List of PWM fault selections.

Values:

enumerator kPWM_Fault_0

Fault 0 input pin

enumerator kPWM_Fault_1

Fault 1 input pin

enumerator kPWM_Fault_2

Fault 2 input pin

enumerator kPWM_Fault_3

Fault 3 input pin

enum __pwm_fault_disable

List of PWM fault disable mapping selections.

Values:

enumerator kPWM_FaultDisable_0

Fault 0 disable mapping

enumerator kPWM_FaultDisable_1

Fault 1 disable mapping

enumerator kPWM_FaultDisable_2

Fault 2 disable mapping

enumerator kPWM_FaultDisable_3

Fault 3 disable mapping

enum __pwm_fault_channels

List of PWM fault channels.

Values:

enumerator kPWM_faultchannel_0

enum __pwm_input_capture_edge

PWM capture edge select.

Values:

enumerator kPWM_Disable

Disabled

enumerator kPWM_FallingEdge

Capture on falling edge only

enumerator kPWM_RisingEdge

Capture on rising edge only

enumerator kPWM_RiseAndFallEdge

Capture on rising or falling edge

enum __pwm_force_signal

PWM output options when a FORCE_OUT signal is asserted.

Values:

enumerator kPWM_UsePwm

Generated PWM signal is used by the deadtime logic.

enumerator kPWM_InvertedPwm

Inverted PWM signal is used by the deadtime logic.

enumerator kPWM_SoftwareControl

Software controlled value is used by the deadtime logic.

enumerator kPWM_UseExternal

PWM_EXT_A signal is used by the deadtime logic.

enum __pwm_chnl_pair_operation

Options available for the PWM A & B pair operation.

Values:

enumerator kPWM_Independent

PWM A & PWM B operate as 2 independent channels

enumerator kPWM_ComplementaryPwmA

PWM A & PWM B are complementary channels, PWM A generates the signal

enumerator kPWM_ComplementaryPwmB

PWM A & PWM B are complementary channels, PWM B generates the signal

enum __pwm_register_reload

Options available on how to load the buffered-registers with new values.

Values:

enumerator kPWM_ReloadImmediate

Buffered-registers get loaded with new values as soon as LDOK bit is set

enumerator kPWM_ReloadPwmHalfCycle

Registers loaded on a PWM half cycle

enumerator kPWM__ReloadPwmFullCycle

Registers loaded on a PWM full cycle

enumerator kPWM__ReloadPwmHalfAndFullCycle

Registers loaded on a PWM half & full cycle

enum __pwm_fault_recovery_mode

Options available on how to re-enable the PWM output when recovering from a fault.

Values:

enumerator kPWM__NoRecovery

PWM output will stay inactive

enumerator kPWM__RecoverHalfCycle

PWM output re-enabled at the first half cycle

enumerator kPWM__RecoverFullCycle

PWM output re-enabled at the first full cycle

enumerator kPWM__RecoverHalfAndFullCycle

PWM output re-enabled at the first half or full cycle

enum __pwm_interrupt_enable

List of PWM interrupt options.

Values:

enumerator kPWM__CompareVal0InterruptEnable

PWM VAL0 compare interrupt

enumerator kPWM__CompareVal1InterruptEnable

PWM VAL1 compare interrupt

enumerator kPWM__CompareVal2InterruptEnable

PWM VAL2 compare interrupt

enumerator kPWM__CompareVal3InterruptEnable

PWM VAL3 compare interrupt

enumerator kPWM__CompareVal4InterruptEnable

PWM VAL4 compare interrupt

enumerator kPWM__CompareVal5InterruptEnable

PWM VAL5 compare interrupt

enumerator kPWM__CaptureX0InterruptEnable

PWM capture X0 interrupt

enumerator kPWM__CaptureX1InterruptEnable

PWM capture X1 interrupt

enumerator kPWM__CaptureB0InterruptEnable

PWM capture B0 interrupt

enumerator kPWM__CaptureB1InterruptEnable

PWM capture B1 interrupt

enumerator kPWM__CaptureA0InterruptEnable

PWM capture A0 interrupt

enumerator kPWM__CaptureA1InterruptEnable

PWM capture A1 interrupt

enumerator kPWM__ReloadInterruptEnable
PWM reload interrupt

enumerator kPWM__ReloadErrorInterruptEnable
PWM reload error interrupt

enumerator kPWM__Fault0InterruptEnable
PWM fault 0 interrupt

enumerator kPWM__Fault1InterruptEnable
PWM fault 1 interrupt

enumerator kPWM__Fault2InterruptEnable
PWM fault 2 interrupt

enumerator kPWM__Fault3InterruptEnable
PWM fault 3 interrupt

enum __pwm_status_flags

List of PWM status flags.

Values:

enumerator kPWM__CompareVal0Flag
PWM VAL0 compare flag

enumerator kPWM__CompareVal1Flag
PWM VAL1 compare flag

enumerator kPWM__CompareVal2Flag
PWM VAL2 compare flag

enumerator kPWM__CompareVal3Flag
PWM VAL3 compare flag

enumerator kPWM__CompareVal4Flag
PWM VAL4 compare flag

enumerator kPWM__CompareVal5Flag
PWM VAL5 compare flag

enumerator kPWM__CaptureX0Flag
PWM capture X0 flag

enumerator kPWM__CaptureX1Flag
PWM capture X1 flag

enumerator kPWM__CaptureB0Flag
PWM capture B0 flag

enumerator kPWM__CaptureB1Flag
PWM capture B1 flag

enumerator kPWM__CaptureA0Flag
PWM capture A0 flag

enumerator kPWM__CaptureA1Flag
PWM capture A1 flag

enumerator kPWM__ReloadFlag
PWM reload flag

enumerator kPWM__ReloadErrorFlag

PWM reload error flag

enumerator kPWM__RegUpdatedFlag

PWM registers updated flag

enumerator kPWM__Fault0Flag

PWM fault 0 flag

enumerator kPWM__Fault1Flag

PWM fault 1 flag

enumerator kPWM__Fault2Flag

PWM fault 2 flag

enumerator kPWM__Fault3Flag

PWM fault 3 flag

enum __pwm_dma_enable

List of PWM DMA options.

Values:

enumerator kPWM__CaptureX0DMAEnable

PWM capture X0 DMA

enumerator kPWM__CaptureX1DMAEnable

PWM capture X1 DMA

enumerator kPWM__CaptureB0DMAEnable

PWM capture B0 DMA

enumerator kPWM__CaptureB1DMAEnable

PWM capture B1 DMA

enumerator kPWM__CaptureA0DMAEnable

PWM capture A0 DMA

enumerator kPWM__CaptureA1DMAEnable

PWM capture A1 DMA

enum __pwm_dma_source_select

List of PWM capture DMA enable source select.

Values:

enumerator kPWM__DMARequestDisable

Read DMA requests disabled

enumerator kPWM__DMAWatermarksEnable

Exceeding a FIFO watermark sets the DMA read request

enumerator kPWM__DMALocalSync

A local sync (VAL1 matches counter) sets the read DMA request

enumerator kPWM__DMALocalReload

A local reload (STS[RF] being set) sets the read DMA request

enum __pwm_watermark_control

PWM FIFO Watermark AND Control.

Values:

enumerator kPWM_FIFOWatermarksOR

Selected FIFO watermarks are OR'ed together

enumerator kPWM_FIFOWatermarksAND

Selected FIFO watermarks are AND'ed together

enum __pwm_mode

PWM operation mode.

Values:

enumerator kPWM_SignedCenterAligned

Signed center-aligned

enumerator kPWM_CenterAligned

Unsigned cente-aligned

enumerator kPWM_SignedEdgeAligned

Signed edge-aligned

enumerator kPWM_EdgeAligned

Unsigned edge-aligned

enum __pwm_level_select

PWM output pulse mode, high-true or low-true.

Values:

enumerator kPWM_HighTrue

High level represents “on” or “active” state

enumerator kPWM_LowTrue

Low level represents “on” or “active” state

enum __pwm_fault_state

PWM output fault status.

Values:

enumerator kPWM_PwmFaultState0

Output is forced to logic 0 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState1

Output is forced to logic 1 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState2

Output is tristated.

enumerator kPWM_PwmFaultState3

Output is tristated.

enum __pwm_reload_source_select

PWM reload source select.

Values:

enumerator kPWM_LocalReload

The local reload signal is used to reload registers

enumerator kPWM_MasterReload

The master reload signal (from submodule 0) is used to reload

enum `_pwm_fault_clear`

PWM fault clearing options.

Values:

enumerator `kPWM_Automatic`

Automatic fault clearing

enumerator `kPWM_ManualNormal`

Manual fault clearing with no fault safety mode

enumerator `kPWM_ManualSafety`

Manual fault clearing with fault safety mode

enum `_pwm_module_control`

Options for submodule master control operation.

Values:

enumerator `kPWM_Control_Module_0`

Control submodule 0's start/stop,buffer reload operation

enumerator `kPWM_Control_Module_1`

Control submodule 1's start/stop,buffer reload operation

enumerator `kPWM_Control_Module_2`

Control submodule 2's start/stop,buffer reload operation

enumerator `kPWM_Control_Module_3`

Control submodule 3's start/stop,buffer reload operation

typedef enum `_pwm_submodule` `pwm_submodule_t`

List of PWM submodules.

typedef enum `_pwm_channels` `pwm_channels_t`

List of PWM channels in each module.

typedef enum `_pwm_value_register` `pwm_value_register_t`

List of PWM value registers.

typedef enum `_pwm_clock_source` `pwm_clock_source_t`

PWM clock source selection.

typedef enum `_pwm_clock_prescale` `pwm_clock_prescale_t`

PWM prescaler factor selection for clock source.

typedef enum `_pwm_force_output_trigger` `pwm_force_output_trigger_t`

Options that can trigger a PWM FORCE_OUT.

typedef enum `_pwm_output_state` `pwm_output_state_t`

PWM channel output status.

typedef enum `_pwm_init_source` `pwm_init_source_t`

PWM counter initialization options.

typedef enum `_pwm_load_frequency` `pwm_load_frequency_t`

PWM load frequency selection.

typedef enum `_pwm_fault_input` `pwm_fault_input_t`

List of PWM fault selections.

typedef enum `_pwm_fault_disable` `pwm_fault_disable_t`

List of PWM fault disable mapping selections.

`typedef enum _pwm_fault_channels pwm_fault_channels_t`

List of PWM fault channels.

`typedef enum _pwm_input_capture_edge pwm_input_capture_edge_t`

PWM capture edge select.

`typedef enum _pwm_force_signal pwm_force_signal_t`

PWM output options when a FORCE_OUT signal is asserted.

`typedef enum _pwm_chnl_pair_operation pwm_chnl_pair_operation_t`

Options available for the PWM A & B pair operation.

`typedef enum _pwm_register_reload pwm_register_reload_t`

Options available on how to load the buffered-registers with new values.

`typedef enum _pwm_fault_recovery_mode pwm_fault_recovery_mode_t`

Options available on how to re-enable the PWM output when recovering from a fault.

`typedef enum _pwm_interrupt_enable pwm_interrupt_enable_t`

List of PWM interrupt options.

`typedef enum _pwm_status_flags pwm_status_flags_t`

List of PWM status flags.

`typedef enum _pwm_dma_enable pwm_dma_enable_t`

List of PWM DMA options.

`typedef enum _pwm_dma_source_select pwm_dma_source_select_t`

List of PWM capture DMA enable source select.

`typedef enum _pwm_watermark_control pwm_watermark_control_t`

PWM FIFO Watermark AND Control.

`typedef enum _pwm_mode pwm_mode_t`

PWM operation mode.

`typedef enum _pwm_level_select pwm_level_select_t`

PWM output pulse mode, high-true or low-true.

`typedef enum _pwm_fault_state pwm_fault_state_t`

PWM output fault status.

`typedef enum _pwm_reload_source_select pwm_reload_source_select_t`

PWM reload source select.

`typedef enum _pwm_fault_clear pwm_fault_clear_t`

PWM fault clearing options.

`typedef enum _pwm_module_control pwm_module_control_t`

Options for submodule master control operation.

`typedef struct _pwm_signal_param pwm_signal_param_t`

Structure for the user to define the PWM signal characteristics.

`typedef struct _pwm_config pwm_config_t`

PWM config structure.

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the PWM_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

```
typedef struct _pwm_fault_input_filter_param pwm_fault_input_filter_param_t
```

Structure for the user to configure the fault input filter.

```
typedef struct _pwm_fault_param pwm_fault_param_t
```

Structure is used to hold the parameters to configure a PWM fault.

```
typedef struct _pwm_input_capture_param pwm_input_capture_param_t
```

Structure is used to hold parameters to configure the capture capability of a signal pin.

```
void PWM_SetupInputCapture(PWM_Type *base, pwm_submodule_t subModule,  
                           pwm_channels_t pwmChannel, const  
                           pwm_input_capture_param_t *inputCaptureParams)
```

Sets up the PWM input capture.

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel in the submodule to setup
- inputCaptureParams – Parameters passed in to set up the input pin

```
status_t PWM_GetInputCaptureValue(PWM_Type *base, pwm_submodule_t subModule,  
                                  pwm_channels_t pwmChannel, uint8_t captureIndex,  
                                  uint16_t *captureValue)
```

Read the capture value.

This function reads the capture value stored in channel's capture value register. It should be called when a valid edge is detected on the input capture pin(related capture flag is set). The capture circuit has two input capture registers per channel for first edge and second edge capture.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to read from (PWM A, PWM B, or PWM X)
- captureIndex – Capture register to read (0 for first edge capture, 1 for second edge capture)

Returns

Returns kStatus_InvalidArgument if pwmChannel does not support capture feature; kStatus_Success otherwise

```
void PWM_SetupFaultInputFilter(PWM_Type *base, const pwm_fault_input_filter_param_t  
                              *faultInputFilterParams)
```

Sets up the PWM fault channel 0 input filter.

Parameters

- base – PWM peripheral base address
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

```
void PWM_SetupFaultInputFilterExt(PWM_Type *base, pwm_fault_channels_t faultChannel,  
                                  const pwm_fault_input_filter_param_t  
                                  *faultInputFilterParams)
```

Sets up the PWM fault input filter.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

```
void PWM_SetupFaults(PWM_Type *base, pwm_fault_input_t faultNum, const
                    pwm_fault_param_t *faultParams)
```

Sets up the PWM fault channel 0 protection.

Parameters

- base – PWM peripheral base address
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

```
void PWM_SetupFaultsExt(PWM_Type *base, pwm_fault_channels_t faultChannel,
                       pwm_fault_input_t faultNum, const pwm_fault_param_t
                       *faultParams)
```

Sets up the PWM fault protection.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

```
void PWM_FaultDefaultConfig(pwm_fault_param_t *config)
```

Fill in the PWM fault config struct with the default settings.

The default values are:

```
config->faultClearingMode = kPWM_Automatic;
config->faultLevel = false;
config->enableCombinationalPath = true;
config->recoverMode = kPWM_NoRecovery;
```

Parameters

- config – Pointer to user's PWM fault config structure.

```
void PWM_SetupForceSignal(PWM_Type *base, pwm_submodule_t subModule, pwm_channels_t
                        pwmChannel, pwm_force_signal_t mode)
```

Selects the signal to output on a PWM pin when a FORCE_OUT signal is asserted.

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- mode – Signal to output when a FORCE_OUT is triggered

```
static inline void PWM_SetVALxValue(PWM_Type *base, pwm_submodule_t subModule,  
                                   pwm_value_register_t valueRegister, uint16_t value)
```

Set the PWM VALx registers.

This function allows the user to write value into VAL registers directly. And it will destroying the PWM clock period set by the PWM_SetupPwm()/PWM_SetupPwmPhaseShift() functions. Due to VALx registers are bufferd, the new value will not active 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

2.37 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 kINPUTMUX0_RST_SHIFT_RSTn
INPUTMUX0 reset control

enumerator kFREQME_RST_SHIFT_RSTn
FREQME reset control

enumerator kCTIMER0_RST_SHIFT_RSTn
CTIMER0 reset control

enumerator kCTIMER1_RST_SHIFT_RSTn
CTIMER1 reset control

enumerator kDMA0_RST_SHIFT_RSTn
DMA0 reset control

enumerator kCRC0_RST_SHIFT_RSTn
CRC0 reset control

enumerator kEIM0_RST_SHIFT_RSTn
EIM0 reset control

enumerator kERM0_RST_SHIFT_RSTn
ERM0 reset control

enumerator kFLEXPWM0_RST_SHIFT_RSTn
FLEXPWM0 reset control

enumerator kLPI2C0_RST_SHIFT_RSTn
LPI2C0 reset control

enumerator kLPSPi0_RST_SHIFT_RSTn
LPSPi0 reset control

enumerator kLPUART0_RST_SHIFT_RSTn
LPUART0 reset control

enumerator kLPUART1_RST_SHIFT_RSTn
LPUART1 reset control

enumerator kLPUART2_RST_SHIFT_RSTn
LPUART2 reset control

enumerator kLPUART3_RST_SHIFT_RSTn
LPUART3 reset control

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control

enumerator kCMP0_RST_SHIFT_RSTn
CMP0 reset control

enumerator kOPAMP0_RST_SHIFT_RSTn
OPAMP0 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 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 kTRNG0_RST_SHIFT_RSTn
TRNG0 reset control

enumerator NotAvail_RSTn
No 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

CRC_RSTS

Reset bits for CRC peripheral.

Array initializers with peripheral reset bits

CRC_RSTS_N

CTIMER_RSTS

Reset bits for CTIMER peripheral.

CTIMER_RSTS_N

DMA_RSTS

Reset bits for DMA peripheral.

DMA_RSTS_N

EIM_RSTS

Reset bits for EIM peripheral.

EIM_RSTS_N

ERM_RSTS

Reset bits for ERM peripheral.

ERM_RSTS_N

FLEXPWM_RSTS

Reset bits for FLEXPWM peripheral.

FLEXPWM_RSTS_N

FREQME_RSTS

Reset bits for FREQME peripheral.

FREQME_RSTS_N

GPIO_RSTS

Reset bits for GPIO peripheral.

GPIO_RSTS_N

INPUTMUX_RSTS

Reset bits for INPUTMUX peripheral.

INPUTMUX_RSTS_N

LPADC_RSTS

Reset bits for LPADC peripheral.

LPADC_RSTS_N

LPUART_RSTS

Reset bits for LPUART peripheral.

LPUART_RSTS_N

LPSPi_RSTS

Reset bits for LPSPi peripheral.

LPSPi_RSTS_N

LPI2C_RSTS

Reset bits for LPI2C peripheral.

LPI2C_RSTS_N

OPAMP_RSTS

Reset bits for OPAMP peripheral.

OPAMP_RSTS_N

PORT_RSTS

Reset bits for PORT peripheral.

PORT_RSTS_N

TRNG_RSTS

Reset bits for TRNG peripheral.

TRNG_RSTS_N

2.38 RTC: Real Time Clock

`void RTC_Init(RTC_Type *base, const rtc_config_t *config)`

Ungates the RTC clock and configures the peripheral for basic operation.

This function issues a software reset if the timer invalid flag is set.

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.

`static inline 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->clockOutput = false;
config->wakeupSelect = false;
config->updateMode = false;
config->supervisorAccess = false;
config->compensationInterval = 0;
config->compensationTime = 0;
```

Parameters

- `config` – Pointer to the user's RTC configuration structure.

status_t RTC_SetDatetime(RTC_Type *base, const *rtc_datetime_t* *datetime)

Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

- *base* – RTC peripheral base address
- *datetime* – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the RTC
kStatus_InvalidArgument: Error because the datetime format is incorrect

void RTC_GetDatetime(RTC_Type *base, *rtc_datetime_t* *datetime)

Gets the RTC time and stores it in the given time structure.

Parameters

- *base* – RTC peripheral base address
- *datetime* – Pointer to the structure where the date and time details are stored.

status_t RTC_SetAlarm(RTC_Type *base, const *rtc_datetime_t* *alarmTime)

Sets the RTC alarm time.

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

- *base* – RTC peripheral base address
- *alarmTime* – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm
kStatus_InvalidArgument: Error because the alarm datetime format is incorrect
kStatus_Fail: Error because the alarm time has already passed

void RTC_GetAlarm(RTC_Type *base, *rtc_datetime_t* *datetime)

Returns the RTC alarm time.

Parameters

- *base* – RTC peripheral base address
- *datetime* – Pointer to the structure where the alarm date and time details are stored.

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*

void RTC_DisableInterrupts(RTC_Type *base, *uint32_t* mask)

Disables the selected RTC interrupts.

Parameters

- *base* – RTC peripheral base address

- mask – The interrupts to enable. This is a logical OR of members of the enumeration `rtc_interrupt_enable_t`

`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`

`uint32_t RTC_GetStatusFlags(RTC_Type *base)`

Gets the RTC status flags.

Parameters

- base – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `rtc_status_flags_t`

`void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)`

Clears the RTC status flags.

Parameters

- base – RTC peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `rtc_status_flags_t`

`static inline void RTC_EnableLPOClock(RTC_Type *base, bool enable)`

Enable/Disable RTC 1kHz LPO clock.

Note: After setting this bit, RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable RTC 1kHz LPO clock

`static inline void RTC_StartTimer(RTC_Type *base)`

Starts the RTC time counter.

After calling this function, the timer counter increments once a second provided `SR[TOF]` or `SR[TIF]` are not set.

Parameters

- base – RTC peripheral base address

`static inline void RTC_StopTimer(RTC_Type *base)`

Stops the RTC time counter.

RTC's seconds register can be written to only when the timer is stopped.

Parameters

- base – RTC peripheral base address

`void RTC_GetMonotonicCounter(RTC_Type *base, uint64_t *counter)`

Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.

Parameters

- `base` – RTC peripheral base address
- `counter` – Pointer to variable where the value is stored.

`void RTC_SetMonotonicCounter(RTC_Type *base, uint64_t counter)`

Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value. The Monotonic Overflow Flag in `RTC_SR` is cleared due to the API.

Parameters

- `base` – RTC peripheral base address
- `counter` – Counter value

`status_t RTC_IncrementMonotonicCounter(RTC_Type *base)`

Increments the Monotonic Counter by one.

Increments the Monotonic Counter (registers `RTC_MCLR` and `RTC_MCHR` accordingly) by setting the monotonic counter enable (`MER[MCE]`) and then writing to the `RTC_MCLR` register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

Parameters

- `base` – RTC peripheral base address

Returns

`kStatus_Success`: success
`kStatus_Fail`: error occurred, either time invalid or monotonic overflow flag was found

`FSL_RTC_DRIVER_VERSION`

Version 2.4.0

`enum _rtc_interrupt_enable`

List of RTC interrupts.

Values:

enumerator `kRTC_TimeInvalidInterruptEnable`

Time invalid interrupt.

enumerator `kRTC_TimeOverflowInterruptEnable`

Time overflow interrupt.

enumerator `kRTC_AlarmInterruptEnable`

Alarm interrupt.

enumerator `kRTC_MonotonicOverflowInterruptEnable`

Monotonic Overflow Interrupt Enable

enumerator `kRTC_SecondsInterruptEnable`

Seconds interrupt.

enumerator `kRTC_TestModeInterruptEnable`

enumerator `kRTC_FlashSecurityInterruptEnable`

enumerator `kRTC_TamperPinInterruptEnable`

enumerator `kRTC_SecurityModuleInterruptEnable`

enumerator kRTC_LossOfClockInterruptEnable

enum _rtc_status_flags

List of RTC flags.

Values:

enumerator kRTC_TimeInvalidFlag

Time invalid flag

enumerator kRTC_TimeOverflowFlag

Time overflow flag

enumerator kRTC_AlarmFlag

Alarm flag

enumerator kRTC_MonotonicOverflowFlag

Monotonic Overflow Flag

enumerator kRTC_TamperInterruptDetectFlag

Tamper interrupt detect flag

enumerator kRTC_TestModeFlag

enumerator kRTC_FlashSecurityFlag

enumerator kRTC_TamperPinFlag

enumerator kRTC_SecurityTamperFlag

enumerator kRTC_LossOfClockTamperFlag

enum _rtc_osc_cap_load

List of RTC Oscillator capacitor load settings.

Values:

enumerator kRTC_Capacitor_2p

2 pF capacitor load

enumerator kRTC_Capacitor_4p

4 pF capacitor load

enumerator kRTC_Capacitor_8p

8 pF capacitor load

enumerator kRTC_Capacitor_16p

16 pF capacitor load

enum _rtc_timer_seconds_interrupt_frequency

List of RTC Timer Seconds Interrupt Frequencies.

Values:

enumerator kRTC_TimerSecondsFrequency1Hz

Timer seconds frequency is 1Hz

enumerator kRTC_TimerSecondsFrequency2Hz

Timer seconds frequency is 2Hz

enumerator kRTC_TimerSecondsFrequency4Hz

Timer seconds frequency is 4Hz

enumerator kRTC_TimerSecondsFrequency8Hz

Timer seconds frequency is 8Hz

enumerator `kRTC_TimerSecondsFrequency16Hz`

Timer seconds frequency is 16Hz

enumerator `kRTC_TimerSecondsFrequency32Hz`

Timer seconds frequency is 32Hz

enumerator `kRTC_TimerSecondsFrequency64Hz`

Timer seconds frequency is 64Hz

enumerator `kRTC_TimerSecondsFrequency128Hz`

Timer seconds frequency is 128Hz

typedef enum `_rtc_interrupt_enable` `rtc_interrupt_enable_t`

List of RTC interrupts.

typedef enum `_rtc_status_flags` `rtc_status_flags_t`

List of RTC flags.

typedef enum `_rtc_osc_cap_load` `rtc_osc_cap_load_t`

List of RTC Oscillator capacitor load settings.

typedef enum `_rtc_timer_seconds_interrupt_frequency` `rtc_timer_seconds_interrupt_frequency_t`

List of RTC Timer Seconds Interrupt Frequencies.

typedef struct `_rtc_datetime` `rtc_datetime_t`

Structure is used to hold the date and time.

typedef struct `_rtc_pin_config` `rtc_pin_config_t`

RTC pin config structure.

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.

The config struct can be made const so it resides in flash

static inline uint32_t `RTC_GetTamperTimeSeconds(RTC_Type *base)`

Get the RTC tamper time seconds.

Parameters

- `base` – RTC peripheral base address

static inline void `RTC_SetOscCapLoad(RTC_Type *base, uint32_t capLoad)`

This function sets the specified capacitor configuration for the RTC oscillator.

Parameters

- `base` – RTC peripheral base address
- `capLoad` – Oscillator loads to enable. This is a logical OR of members of the enumeration `rtc_osc_cap_load_t`

static inline void `RTC_Reset(RTC_Type *base)`

Performs a software reset on the RTC module.

This resets all RTC registers except for the SWR bit and the `RTC_WAR` and `RTC_RAR` registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

- `base` – RTC peripheral base address

```
static inline void RTC_EnableWakeUpPin(RTC_Type *base, bool enable)
```

Enables or disables the RTC Wakeup Pin Operation.

This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

```
static inline void RTC_EnableClockOutput(RTC_Type *base, bool enable)
```

Enables or disables the RTC 32 kHz clock output.

This function enables or disables the RTC 32 kHz clock output.

Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

```
void RTC_SetTimerSecondsInterruptFrequency(RTC_Type *base,  
                                           rtc_timer_seconds_interrupt_frequency_t freq)
```

Sets the RTC timer seconds interrupt frequency.

This function sets the RTC timer seconds interrupt frequency.

Parameters

- base – RTC peripheral base address
- freq – The timer seconds interrupt frequency. This is a member of the enumeration `rtc_timer_seconds_interrupt_frequency_t`

```
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_pin_config
```

#include <fsl_rtc.h> RTC pin config structure.

Public Members

bool inputLogic

true: Tamper pin input data is logic one. false: Tamper pin input data is logic zero.

bool pinActiveLow

true: Tamper pin is active low. false: Tamper pin is active high.

bool filterEnable

true: Input filter is enabled on the tamper pin. false: Input filter is disabled on the tamper pin.

bool pullSelectNegate

true: Tamper pin pull resistor direction will negate the tamper pin. false: Tamper pin pull resistor direction will assert the tamper pin.

bool pullEnable

true: Pull resistor is enabled on tamper pin. false: Pull resistor is disabled on tamper pin.

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.

The config struct can be made const so it resides in flash

Public Members

bool clockOutput

true: The 32 kHz clock is not output to other peripherals; false: The 32 kHz clock is output to other peripherals

bool wakeupSelect

true: Wakeup pin outputs the 32 KHz clock; false: Wakeup pin used to wakeup the chip

bool updateMode

true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked

bool supervisorAccess

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported

uint32_t compensationInterval

Compensation interval that is written to the CIR field in RTC TCR Register

uint32_t compensationTime

Compensation time that is written to the TCR field in RTC TCR Register

2.39 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(Flash Logical Window) configuration.

```
config->blockCount    = false;
config->arrayBaseAddr = false;
config->lock           = false;
config->enable         = false;
```

Parameters

- flwConfiguration – Pointer to the configuration structure.

void TRDC_SetFlashLogicalWindow(TRDC_Type *base, const *trdc_flw_config_t* *flwConfiguration)

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xFFFFFFFF;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- flwConfiguration – Pointer to the configuration structure.

status_t TRDC_GetAndClearFirstDomainError(TRDC_Type *base, *trdc_domain_error_t* *error)

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.

Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the `kStatus_NoData`.

status_t TRDC_GetAndClearFirstSpecificDomainError(TRDC_Type *base, *trdc_domain_error_t* *error, *uint8_t* domainId)

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.

- domainId – The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the `kStatus_Success`. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the `kStatus_NoData`.

```
static inline void TRDC_SetMrcGlobalValid(TRDC_Type *base)
```

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

```
static inline uint8_t TRDC_GetMrcRegionNumber(TRDC_Type *base, uint8_t mrcIdx)
```

Gets the TRDC MRC(Memory Region Checkers) region number valid.

Parameters

- base – TRDC peripheral base address.

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 domains whose NSE bits to clear.

`void TRDC_MrcSetRegionDescriptorConfig(TRDC_Type *base, const trdc_mrc_region_descriptor_config_t *config)`

Sets the configuration for one of the region descriptor per domain per MRC instance.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to region descriptor configuration structure.

`static inline void TRDC_SetMbcGlobalValid(TRDC_Type *base)`

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

`void TRDC_GetMbcHardwareConfig(TRDC_Type *base, trdc_slave_memory_hardware_config_t *config, uint8_t mbcIdx, uint8_t slvIdx)`

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

```
void TRDC__MbcSetNseUpdateConfig(TRDC_Type *base, const trdc_mbc_nse_update_config_t
                                *config, uint8_t mbcIdx)
```

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be 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.

2.40 Trdc_core

```
typedef struct _TRDC_General_Type TRDC_General_Type
    TRDC general configuration register definition.
typedef struct _TRDC_FLW_Type TRDC_FLW_Type
    TRDC flash logical control register definition.
typedef struct _TRDC_DomainError_Type TRDC_DomainError_Type
    TRDC domain error register definition.
typedef struct _TRDC_DomainAssignment_Type TRDC_DomainAssignment_Type
    TRDC master domain assignment register definition.
typedef struct _TRDC_MBC_Type TRDC_MBC_Type
    TRDC MBC control register definition.
typedef struct _TRDC_MRC_Type TRDC_MRC_Type
    TRDC MRC control register definition. MRC_DOM0_RGD_W[region][word].
TRDC_GENERAL_BASE(base)
    TRDC base address convert macro.
TRDC_FLW_BASE(base)
TRDC_DOMAIN_ERROR_BASE(base)
TRDC_DOMAIN_ASSIGNMENT_BASE(base)
TRDC_MBC_BASE(base, instance)
TRDC_MRC_BASE(base, instance)
struct _TRDC_General_Type
    #include <fsl_trdc_core.h> TRDC general configuration register definition.
```

Public Members

```
__IO uint32_t TRDC_CR
    TRDC Register, offset: 0x0
__I uint32_t TRDC_HWCFG0
    TRDC Hardware Configuration Register 0, offset: 0xF0
__I uint32_t TRDC_HWCFG1
    TRDC Hardware Configuration Register 1, offset: 0xF4
__I uint32_t TRDC_HWCFG2
    TRDC Hardware Configuration Register 2, offset: 0xF8
__I uint32_t TRDC_HWCFG3
    TRDC Hardware Configuration Register 3, offset: 0xFC
__I uint8_t DACFG [8]
    Domain Assignment Configuration Register, array offset: 0x100, array step: 0x1
__IO uint32_t TRDC_IDAU_CR
    TRDC IDAU Control Register, offset: 0x1C0
struct _TRDC_FLW_Type
    #include <fsl_trdc_core.h> TRDC flash logical control register definition.
```

Public Members

__IO uint32_t TRDC_FLW_CTL
TRDC FLW Control, offset: 0x1E0

__I uint32_t TRDC_FLW_PBASE
TRDC FLW Physical Base, offset: 0x1E4

__IO uint32_t TRDC_FLW_ABASE
TRDC FLW Array Base, offset: 0x1E8

__IO uint32_t TRDC_FLW_BCNT
TRDC FLW Block Count, offset: 0x1EC

struct _TRDC_DomainError_Type

#include <fsl_trdc_core.h> TRDC domain error register definition.

Public Members

__IO uint32_t TRDC_FDID
TRDC Fault Domain ID, offset: 0x1FC

__I uint32_t TRDC_DERRLOC [16]
TRDC Domain Error Location Register, array offset: 0x200, array step: 0x4

struct _TRDC_DomainAssignment_Type

#include <fsl_trdc_core.h> TRDC master domain assignment register definition.

Public Members

__IO uint32_t PID [8]
Process Identifier, array offset: 0x700, array step: 0x4

struct _TRDC_MBC_Type

#include <fsl_trdc_core.h> TRDC MBC control register definition.

Public Members

__I uint32_t MBC_MEM_GLBCFG [4]
MBC Global Configuration Register, array offset: 0x10000, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_NSE_BLK_INDEX
MBC NonSecure Enable Block Index, array offset: 0x10010, array step: 0x2000

__O uint32_t MBC_NSE_BLK_SET
MBC NonSecure Enable Block Set, array offset: 0x10014, array step: 0x2000

__O uint32_t MBC_NSE_BLK_CLR
MBC NonSecure Enable Block Clear, array offset: 0x10018, array step: 0x2000

__O uint32_t MBC_NSE_BLK_CLR_ALL
MBC NonSecure Enable Block Clear All, array offset: 0x1001C, array step: 0x2000

__IO uint32_t MBC_MEMN_GLBAC [8]
MBC Global Access Control, array offset: 0x10020, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM0_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10340, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10380, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103D0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM1_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10640, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10740, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10780, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107C8, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM3_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10840, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10940, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10980, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10A40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BA8, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM5_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10D80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10E40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10F40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10F80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FA0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM7_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11340, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11380, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM9_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM11_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11640, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM11_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11740, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM11_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11780, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM11_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM11_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM11_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM11_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM11_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11840, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11940, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11980, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11A40, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM13_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11D80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11DA0, array step: 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

___I uint32_t MRC_GLBCFG
MRC Global Configuration Register, array offset: 0x14000, array step: 0x1000

___IO uint32_t MRC_NSE_RGN_INDIRECT
MRC NonSecure Enable Region Indirect, array offset: 0x14010, array step: 0x1000

___O uint32_t MRC_NSE_RGN_SET
MRC NonSecure Enable Region Set, array offset: 0x14014, array step: 0x1000

___O uint32_t MRC_NSE_RGN_CLR
MRC NonSecure Enable Region Clear, array offset: 0x14018, array step: 0x1000

___O uint32_t MRC_NSE_RGN_CLR_ALL
MRC NonSecure Enable Region Clear All, array offset: 0x1401C, array step: 0x1000

___IO uint32_t MRC_GLBAC [8]
MRC Global Access Control, array offset: 0x14020, array step: index*0x1000, index2*0x4

___IO uint32_t MRC_DOM0_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14040, array step: index*0x1000, index2*0x8, index3*0x4

```

__IO uint32_t MRC_DOM0_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x140C0, array step: 0x1000
__IO uint32_t MRC_DOM1_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14140,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM1_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x141C0, array step: 0x1000
__IO uint32_t MRC_DOM2_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14240,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM2_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x142C0, array step: 0x1000
__IO uint32_t MRC_DOM3_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14340,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM3_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x143C0, array step: 0x1000
__IO uint32_t MRC_DOM4_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14440,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM4_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x144C0, array step: 0x1000
__IO uint32_t MRC_DOM5_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14540,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM5_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x145C0, array step: 0x1000
__IO uint32_t MRC_DOM6_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14640,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM6_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x146C0, array step: 0x1000
__IO uint32_t MRC_DOM7_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14740,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM7_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x147C0, array step: 0x1000
__IO uint32_t MRC_DOM8_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14840,
    array step: index*0x1000, index2*0x8, index3*0x4
__IO uint32_t MRC_DOM8_RGD_NSE
    MRC Region Descriptor NonSecure Enable, array offset: 0x148C0, array step: 0x1000
__IO uint32_t MRC_DOM9_RGD_W [16][2]
    MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14940,
    array step: index*0x1000, index2*0x8, index3*0x4

```

___IO uint32_t MRC_DOM9_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x149C0, array step: 0x1000

___IO uint32_t MRC_DOM10_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14A40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM10_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14AC0, array step: 0x1000

___IO uint32_t MRC_DOM11_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14B40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM11_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14BC0, array step: 0x1000

___IO uint32_t MRC_DOM12_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14C40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM12_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14CC0, array step: 0x1000

___IO uint32_t MRC_DOM13_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14D40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM13_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14DC0, array step: 0x1000

___IO uint32_t MRC_DOM14_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14E40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM14_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14EC0, array step: 0x1000

___IO uint32_t MRC_DOM15_RGD_W [16][2]
MRC Region Descriptor Word 0..MRC Region Descriptor Word 1, array offset: 0x14F40, array step: index*0x1000, index2*0x8, index3*0x4

___IO uint32_t MRC_DOM15_RGD_NSE
MRC Region Descriptor NonSecure Enable, array offset: 0x14FC0, array step: 0x1000

struct MBC_DERR

Public Members

___I uint32_t W0
MBC Domain Error Word0 Register, array offset: 0x400, array step: 0x10

___I uint32_t W1
MBC Domain Error Word1 Register, array offset: 0x404, array step: 0x10

___O uint32_t W3
MBC Domain Error Word3 Register, array offset: 0x40C, array step: 0x10

struct MRC_DERR

Public Members

```

__I uint32_t W0
    MRC Domain Error Word0 Register, array offset: 0x480, array step: 0x10
__I uint32_t W1
    MRC Domain Error Word1 Register, array offset: 0x484, array step: 0x10
__O uint32_t W3
    MRC Domain Error Word3 Register, array offset: 0x48C, array step: 0x10
union __unnamed33__

```

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

```

2.41 Trdc_soc

FSL_TRDC_SOC_DRIVER_VERSION

Driver version 2.0.0.

TRDC_MBC_MEM_GLBCFG_NBLKS_MASK

TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_MASK

TRDC_MBC_MEM_GLBCFG_SIZE_LOG2_SHIFT

TRDC_MBC_NSE_BLK_CLR_ALL_MEMSEL(x)

TRDC_MBC_NSE_BLK_CLR_ALL_DID_SEL(x)

FSL_FEATURE_TRDC_DOMAIN_COUNT

TRDC feature.

TRDC_MBC_COUNT

TRDC base address convert macro.

TRDC_MBC_OFFSET(x)

MBC register offset in TRDC_Type structure.

TRDC_MBC_ARRAY_STEP

Offset between two MBC control block, useless if there is only one.

FSL_COMPONENT_ID

2.42 TRNG: True Random Number Generator

FSL_TRNG_DRIVER_VERSION

TRNG driver version 2.0.21.

Current version: 2.0.21

Change log:

- version 2.0.21
 - Added support for MCXC devices.
- version 2.0.20
 - Added support for MCXA devices.
- version 2.0.19
 - Added support for MCXA and MCXL.
- version 2.0.18
 - TRNG health checks now done in software on RT5xx and RT6xx.
- version 2.0.17
 - Added support for RT700.
- version 2.0.16
 - Added support for Dual oscillator mode.
- version 2.0.15
 - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest recommended by design team.
- version 2.0.14
 - add support for RW610 and RW612
- version 2.0.13
 - After deepsleep it might return error, added clearing bits in TRNG_GetRandomData() and generating new entropy.
 - Modified reloading entropy in TRNG_GetRandomData(), for some data length it doesn't reloading entropy correctly.
- version 2.0.12
 - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.11
 - Add clearing pending errors in TRNG_Init().
- version 2.0.10
 - Fixed doxygen issues.

- version 2.0.9
 - Fix HIS_CCM metrics issues.
- version 2.0.8
 - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.
- version 2.0.7
 - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
 - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.5
 - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
 - Fix MISRA-2012 issues.
- Version 2.0.3
 - update TRNG_Init to restart entropy generation
- Version 2.0.2
 - fix MISRA issues
- Version 2.0.1
 - add support for KL8x and KL28Z
 - update default OSCDIV for K81 to divide by 2

enum _trng_sample_mode

TRNG sample mode. Used by trng_config_t.

Values:

enumerator kTRNG_SampleModeVonNeumann

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeRaw

Use raw data into both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeVonNeumannRaw

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum _trng_clock_mode

TRNG clock mode. Used by trng_config_t.

Values:

enumerator kTRNG_ClockModeRingOscillator

Ring oscillator is used to operate the TRNG (default).

enumerator kTRNG_ClockModeSystem

System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

enum *trng_ring_osc_div*

TRNG ring oscillator divide. Used by *trng_config_t*.

Values:

enumerator *kTRNG_RingOscDiv0*

Ring oscillator with no divide

enumerator *kTRNG_RingOscDiv2*

Ring oscillator divided-by-2.

enumerator *kTRNG_RingOscDiv4*

Ring oscillator divided-by-4.

enumerator *kTRNG_RingOscDiv8*

Ring oscillator divided-by-8.

enum *trng_oscillator_mode_t*

TRNG oscillator mode . Used by *trng_config_t*.

Values:

enumerator *kTRNG_SingleOscillatorModeOsc1*

Single oscillator mode, using OSC1 (default)

enumerator *kTRNG_DualOscillatorMode*

Dual oscillator mode

enumerator *kTRNG_SingleOscillatorModeOsc2*

Single oscillator mode, using OSC2

typedef enum *trng_sample_mode* *trng_sample_mode_t*

TRNG sample mode. Used by *trng_config_t*.

typedef enum *trng_clock_mode* *trng_clock_mode_t*

TRNG clock mode. Used by *trng_config_t*.

typedef enum *trng_ring_osc_div* *trng_ring_osc_div_t*

TRNG ring oscillator divide. Used by *trng_config_t*.

typedef enum *trng_oscillator_mode_t* *trng_oscillator_mode_t*

TRNG oscillator mode . Used by *trng_config_t*.

typedef struct *trng_statistical_check_limit* *trng_statistical_check_limit_t*

Data structure for definition of statistical check limits. Used by *trng_config_t*.

typedef struct *trng_user_config* *trng_config_t*

Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the *TRNG_Init()* function. It contains all TRNG configurations.

status_t *TRNG_GetDefaultConfig(trng_config_t *userConfig)*

Initializes the user configuration structure to default values.

This function initializes the configuration structure to default values. The default values are platform dependent.

Parameters

- *userConfig* – User configuration structure.

Returns

If successful, returns the *kStatus_TRNG_Success*. Otherwise, it returns an error.

status_t TRNG_Init(TRNG_Type *base, const *trng_config_t* *userConfig)

Initializes the TRNG.

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

- base – TRNG base address
- userConfig – Pointer to the initialization configuration structure.

Returns

If successful, returns the *kStatus_TRNG_Success*. Otherwise, it returns an error.

void TRNG_Deinit(TRNG_Type *base)

Shuts down the TRNG.

This function shuts down the TRNG.

Parameters

- base – TRNG base address.

status_t TRNG_GetRandomData(TRNG_Type *base, void *data, *size_t* dataSize)

Gets random data.

This function gets random data from the TRNG.

Parameters

- base – TRNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

Returns

random data

struct *_trng_statistical_check_limit*

#include <fsl_trng.h> Data structure for definition of statistical check limits. Used by *trng_config_t*.

Public Members

uint32_t maximum

Maximum limit.

int32_t minimum

Minimum limit.

struct *_trng_user_config*

#include <fsl_trng.h> Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the TRNG_Init() function. It contains all TRNG configurations.

Public Members

bool lock

Disable programmability of TRNG registers.

trng_clock_mode_t clockMode

Clock mode used to operate TRNG.

trng_ring_osc_div_t ringOscDiv

Ring oscillator divide used by TRNG.

trng_sample_mode_t sampleMode

Sample mode of the TRNG ring oscillator.

trng_oscillator_mode_t oscillatorMode

TRNG oscillator mode .

trng_ring_osc_div_t ringOsc2Div

Divider used for Ring oscillator 2.

uint16_t entropyDelay

Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken.

uint16_t sampleSize

Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation.

uint16_t sparseBitLimit

Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated. This limit is used only for during von Neumann sampling (enabled by `TRNG_HAL_SetSampleMode()`). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

uint8_t retryCount

Retry count. It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

uint8_t longRunMaxLimit

Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

trng_statistical_check_limit_t monobitLimit

Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.

trng_statistical_check_limit_t runBit1Limit

Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.

trng_statistical_check_limit_t runBit2Limit

Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.

trng_statistical_check_limit_t runBit3Limit

Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.

trng_statistical_check_limit_t runBit4Limit

Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.

trng_statistical_check_limit_t runBit5Limit

Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.

trng_statistical_check_limit_t runBit6PlusLimit

Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.

trng_statistical_check_limit_t pokerLimit

Maximum and minimum limits for statistical check of “Poker Test”.

trng_statistical_check_limit_t frequencyCountLimit

Maximum and minimum limits for statistical check of entropy sample frequency count.

2.43 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.

```
static inline uint32_t WUU_GetModuleInterruptFlag(WUU_Type *base)
```

Get wakeup flags for internal wakeup modules.

Parameters

- `base` – WUU peripheral base address.

Returns

Wakeup flags for all internal wakeup modules.

```
static inline bool WUU_GetInternalWakeupModuleFlag(WUU_Type *base, uint32_t moduleIndex)
```

Gets the internal module wakeup source flag.

This function checks the flag to detect whether the system is woken up by specific on-chip module interrupt.

Parameters

- `base` – WWU peripheral base address.
- `moduleIndex` – A module index, which starts from 0.

Returns

True if the specific pin is a wake up source.

```
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 2.4.2.

```
enum _wu_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 *wuu_external_pin_edge_detection* wuu_external_pin_edge_detection_t

External WakeUp pin edge detection enumeration.

typedef enum *wuu_external_wakeup_pin_event* wuu_external_wakeup_pin_event_t

External input wake up pin event enumeration.

typedef enum *wuu_external_wakeup_pin_mode* wuu_external_wakeup_pin_mode_t

External input wake up pin mode enumeration.

typedef enum *wuu_internal_wakeup_module_event* wuu_internal_wakeup_module_event_t

Internal module wake up event enumeration.

typedef enum *wuu_filter_edge* wuu_filter_edge_t

Pin filter edge enumeration.

typedef enum *wuu_filter_event* wuu_filter_event_t

Pin Filter event enumeration.

typedef enum *wuu_filter_mode* wuu_filter_mode_t

Pin filter mode enumeration.

typedef struct *wuu_external_wakeup_pin_config* wuu_external_wakeup_pin_config_t

External WakeUp pin configuration.

typedef struct *wuu_pin_filter_config* wuu_pin_filter_config_t

Pin Filter configuration.

struct *wuu_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.

wwu_filter_edge_t edge

The edge of the pin digital filter.

wwu_filter_event_t event

The event of the filter output.

wwu_filter_mode_t mode

The mode of the filter operate.

2.44 WWDT: Windowed Watchdog Timer Driver

`void WWDT_GetDefaultConfig(wwdt_config_t *config)`

Initializes WWDT configure structure.

This function initializes the WWDT configure structure to default value. The default value are:

```
config->enableWwdt = true;
config->enableWatchdogReset = false;
config->enableWatchdogProtect = false;
config->enableLockOscillator = false;
config->windowValue = 0xFFFFFfU;
config->timeoutValue = 0xFFFFFfU;
config->warningValue = 0;
```

See also:

`wwdt_config_t`

Parameters

- `config` – Pointer to WWDT config structure.

`void WWDT_Init(WWDT_Type *base, const wwdt_config_t *config)`

Initializes the WWDT.

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
wwdt_config_t config;
WWDT_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
WWDT_Init(wwdt_base,&config);
```

Parameters

- `base` – WWDT peripheral base address
- `config` – The configuration of WWDT

`void WWDT_Deinit(WWDT_Type *base)`

Shuts down the WWDT.

This function shuts down the WWDT.

Parameters

- `base` – WWDT peripheral base address

```
static inline void WWDT_Enable(WWDT_Type *base)
```

Enables the WWDT module.

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

- base – WWDT peripheral base address

```
static inline void WWDT_Disable(WWDT_Type *base)
```

Disables the WWDT module.

Deprecated:

Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

- base – WWDT peripheral base address

```
static inline uint32_t WWDT_GetStatusFlags(WWDT_Type *base)
```

Gets all WWDT status flags.

This function gets all status flags.

Example for getting Timeout Flag:

```
uint32_t status;
status = WWDT_GetStatusFlags(wwdt_base) & kWWDT_TimeoutFlag;
```

Parameters

- base – WWDT peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `_wwdt_status_flags_t`

```
void WWDT_ClearStatusFlags(WWDT_Type *base, uint32_t mask)
```

Clear WWDT flag.

This function clears WWDT status flag.

Example for clearing warning flag:

```
WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
```

Parameters

- base – WWDT peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `_wwdt_status_flags_t`

```
static inline void WWDT_SetWarningValue(WWDT_Type *base, uint32_t warningValue)
```

Set the WWDT warning value.

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

- base – WWDT peripheral base address
- warningValue – WWDT warning value.

static inline void WWDT_SetTimeoutValue(WWDT_Type *base, uint32_t timeoutCount)

Set the WWDT timeout value.

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is $TWDCLK * 256 * 4$. If enableWatchdogProtect flag is true in `wwdt_config_t` config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

- base – WWDT peripheral base address
- timeoutCount – WWDT timeout value, count of WWDT clock tick.

static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)

Sets the WWDT window value.

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFF (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

```
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 Boot

3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource

Overview

This repository is a fork of MCUboot (<https://github.com/mcu-tools/mcuboot>) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation

Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [MCUboot - Documentation](#) to review details on the contents in this sub-repo.

Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

Contribution

Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream MCUboot project. Once this MCUboot fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

NXP Fork

This fork of MCUboot contains specific modifications and enhancements for NXP MCUXpresso SDK integration.

See *changelog* for details.

3.1.2 MCUboot



This is MCUboot version 2.3.0

MCUboot is a secure bootloader for 32-bits microcontrollers. It defines a common infrastructure for the bootloader and the system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software upgrade.

MCUboot is not dependent on any specific operating system and hardware and relies on hardware porting layers from the operating system it works with. Currently, MCUboot works with the following operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

RIOT is supported only as a boot target. We will accept any new port contributed by the community once it is good enough.

MCUboot How-tos

See the following pages for instructions on using MCUboot with different operating systems and SoCs:

- [Zephyr](#)
- [Apache Mynewt](#)
- [Apache NuttX](#)
- [RIOT](#)
- [Mbed OS](#)
- [Espressif](#)
- [Cypress/Infineon](#)

There are also instructions for the *Simulator*.

Roadmap

The issues being planned and worked on are tracked using GitHub issues. To give your input, visit [MCUboot GitHub Issues](#).

Source files

You can find additional documentation on the bootloader in the source files. For more information, use the following links:

- [boot/bootutil](#) - The core of the bootloader itself.
- [boot/boot_serial](#) - Support for serial upgrade within the bootloader itself.
- [boot/zephyr](#) - Port of the bootloader to Zephyr.
- [boot/mynewt](#) - Bootloader application for Apache Mynewt.
- [boot/nuttX](#) - Bootloader application and port of MCUboot interfaces for Apache NuttX.
- [boot/mbed](#) - Port of the bootloader to Mbed OS.
- [boot/espressif](#) - Bootloader application and MCUboot port for Espressif SoCs.
- [boot/cypress](#) - Bootloader application and MCUboot port for Cypress/Infineon SoCs.
- [imgtool](#) - A tool to securely sign firmware images for booting by MCUboot.
- [sim](#) - A bootloader simulator for testing and regression.

Joining the project

Developers are welcome!

Use the following links to join or see more about the project:

- [Our developer mailing list](#)
- [Our Discord channel](#) [Get your invite](#)

3.2 File System

3.2.1 FatFs

MCUXpresso SDK : mcuxsdk-middleware-fatfs

Overview This repository is for FatFs 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 [FatFs - Documentation](#) to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest [Getting Started with SDK - Detailed Installation Instructions](#)

Contribution Contributions are not currently accepted. Guidelines to contribute will be posted in the future.

Repo Specific Content This is MCUXpresso SDK fork of FatFs (FAT file system created by ChaN). Official documentation is available at <http://elm-chan.org/fsw/ff/>

MCUXpresso version is extending original content by following hardware specific porting layers:

- mmc_disk
- nand_disk
- ram_disk
- sd_disk
- sdspi_disk
- usb_disk

Changelog FatFs

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#)

[R0.16_rev0]

- Upgraded to version 0.16
- Applied patch <https://elm-chan.org/fsw/ff/patch/ff16p1.diff>
- Applied patch <https://elm-chan.org/fsw/ff/patch/ffunicode.zip>
- Make lb and hb variables conditional. Fix ‘unused variable’ issue:
 - ffunicode.c:10622:28: error: unused variable ‘lb’
 - ffunicode.c:10622:7: error: unused variable ‘hb’

[R0.15_rev0]

- Upgraded to version 0.15
- Applied patches from <http://elm-chan.org/fsw/ff/patches.html>

[R0.14b_rev1]

- Applied patches from <http://elm-chan.org/fsw/ff/patches.html>

[R0.14b_rev0]

- Upgraded to version 0.14b

[R0.14a_rev0]

- Upgraded to version 0.14a
- Applied patch ff14a_p1.diff and ff14a_p2.diff

[R0.14_rev0]

- Upgraded to version 0.14
- Applied patch ff14_p1.diff and ff14_p2.diff

[R0.13c_rev0]

- Upgraded to version 0.13c
- Applied patches ff_13c_p1.diff, ff_13c_p2.diff, ff_13c_p3.diff and ff_13c_p4.diff.

[R0.13b_rev0]

- Upgraded to version 0.13b

[R0.13a_rev0]

- Upgraded to version 0.13a. Added patch ff_13a_p1.diff.

[R0.12c_rev1]

- Add NAND disk support.

[R0.12c_rev0]

- Upgraded to version 0.12c and applied patches ff_12c_p1.diff and ff_12c_p2.diff.

[R0.12b_rev0]

- Upgraded to version 0.12b.

[R0.11a]

- Added glue functions for low-level drivers (SDHC, SDSPI, RAM, MMC). Modified diskio.c.
- Added RTOS wrappers to make FatFs thread safe. Modified syscall.c.
- Renamed ffconf.h to ffconf_template.h. Each application should contain its own ffconf.h.
- Included ffconf.h into diskio.c to enable the selection of physical disk from ffconf.h by macro definition.
- Conditional compilation of physical disk interfaces in diskio.c.

3.3 Motor Control

3.3.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called “packet-driven BDM” interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to “packet-driven BDM”, the FreeMASTER also supports a communication over [J-Link RTT](<https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/>) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to [FreeMASTER community](#) or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the `src/platforms` directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called `FMSTR_TRANSPORT` with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The `mcuxsdk` folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “`ampsdk`” drivers target automotive-specific MCUs and their respective SDKs. The “`dreg`” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the `FMSTR_Init` function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a “middleware” component which may be downloaded along with the example applications from <https://mcuxpresso.nxp.com/en/welcome>.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- [The official FreeMASTER middleware repository.](#)
- [Online version of this document](#)

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer’s physical or virtual COM port. The typical transmission speed is 115200 bps.
- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer’s USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM,

the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.

- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER “middleware” driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample applications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.

- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- “External” single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- “True” single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as “automatic” in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- **src/platforms** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- **src/common** folder—contains the common driver source files shared by the driver for all supported platforms. All the .c files must be added to the project, compiled, and linked together with the application.

- *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
- *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
- *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
- *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
- *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
- *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
- *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
- *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster_can.h* - defines the low-level message-oriented CAN API.
- *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.

- *freemaster_net.h* - definitions related to the Network transport.
- *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster_utils.h* - definitions related to utility code.
- **src/drivers/[sdk]/serial** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
 - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- **src/drivers/[sdk]/can** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
 - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- **src/drivers/[sdk]/network** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
 - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
 - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR    [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See [Driver interrupt modes](#).

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- FMSTR_SERIAL - serial communication protocol
- FMSTR_CAN - using CAN communication
- FMSTR_PDBDM - using packet-driven BDM communication
- FMSTR_NET - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- FMSTR_SERIAL_MCUX_UART - UART driver
- FMSTR_SERIAL_MCUX_LPUART - LPUART driver
- FMSTR_SERIAL_MCUX_USART - USART driver
- FMSTR_SERIAL_MCUX_MINIUSART - miniUSART driver
- FMSTR_SERIAL_MCUX_QSCI - QSCI driver
- FMSTR_SERIAL_MCUX_USB - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)

- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetSerialBaseAddress() to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode. The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.

Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.

The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options**FMSTR_USE_RECORDER**

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()' API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()' API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using `FMSTR_RegisterAppCmdCall()`. Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool. Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables. Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project. Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the `FMSTR_SetUpTsaBuff()` and `FMSTR_TsaAddVar()` functions. Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_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_QUEUE_SIZE` macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);  
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see [Driver interrupt modes](#)), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the `FMSTR_USE_RECORDER` configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by `FMSTR_Init` when the *freemaster_cfg.h* configuration file defines the `FMSTR_REC_BUFF_SIZE` and `FMSTR_REC_TIMEBASE` options.

For more information, see [Configurable items](#).

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the *FMSTR_USE_TSA* macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must 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_UINT n	Unsigned integer type of size n bits ($n=8,16,32,64$)
FMSTR_TSA_SINT n	Signed integer type of size n bits ($n=8,16,32,64$)
FMSTR_TSA_FRAC n	Fractional number of size n bits ($n=16,32,64$).
FMSTR_TSA_FRAC_Q(m,n)	Signed fractional number in general Q form ($m+n+1$ total bits)
FMSTR_TSA_FRAC_UQ(m,n)	Unsigned fractional number in general UQ form ($m+n$ total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FM-STR_TSA_USERTYPE($name$)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")
```

(continues on next page)

(continued from previous page)

```
/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR ↵
↵ tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object

- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
 - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
 - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
 - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the *FMSTR_USE_TSA_DYNAMIC* configuration option and when the *FMSTR_SetUpTsaBuff* function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the *FMSTR_APPCMDRESULT_NOCMD* constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the *FMSTR_AppCmdAck* call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The *FMSTR_GetAppCmd* function does not report the commands for which a callback handler function exists. If the *FMSTR_GetAppCmd* function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns *FMSTR_APPCMDRESULT_NOCMD*.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↳ PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↳
FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_XXX and FMSTR_PIPE_SIZE_XXX constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the *nGranularity* value equal to the *nLength* value, all data are considered as one chunk which is either written successfully as a whole or not at all. The *nGranularity* value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The `readGranularity` argument can be used to copy the data in larger chunks in the same way as described in the `FMSTR_PipeWrite` function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
<i>FM-STR_ADDR</i> For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
<i>FM-STR_SIZE</i> It is required that this type is unsigned and at least 16 bits wide integer.	Data type used to hold the memory block size.
<i>FM-STR_BOOL</i> This type is used only in zero/non-zero conditions in the driver code.	Data type used as a general boolean type.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command code.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to create the Application Command data buffer.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command result code.

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

<i>FM-STR_TSA_TII</i>	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables. By default, this is defined as <i>FM-STR_SIZE</i> .
<i>FM-STR_TSA_TS</i>	Data type used to hold a memory block size, as used in the TSA descriptors. By default, this is defined as <i>FM-STR_SIZE</i> .

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

<i>FM-STR_HPIPE</i>	Pipe handle that identifies the open-pipe object. Generally, this is a pointer to a void type.
<i>FM-STR_PIPE_PC</i>	Integer type required to hold at least 7 bits of data. Generally, this is an unsigned 8-bit or 16-bit type.
<i>FM-STR_PIPE_SI</i>	Integer type required to hold at least 16 bits of data. This is used to store the data buffer sizes.
<i>FM-STR_PPIPEF</i>	Pointer to the pipe handler function. See FM-STR_PipeOpen for more details.

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

<i>FMSTR_U8</i>	The smallest memory entity.
On the vast majority of platforms, this is an unsigned 8-bit integer.	
On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.	
<i>FM-STR_U16</i>	Unsigned 16-bit integer.
<i>FM-STR_U32</i>	Unsigned 32-bit integer.
<i>FMSTR_S8</i>	Signed 8-bit integer.
<i>FM-STR_S16</i>	Signed 16-bit integer.
<i>FM-STR_S32</i>	Signed 32-bit integer.
<i>FM-STR_FLOAT</i>	4-byte standard IEEE floating-point type.
<i>FM-STR_FLAGS</i>	Data type forming a union with a structure of flag bit-fields.
<i>FM-STR_SIZE8</i>	Data type holding a general size value, at least 8 bits wide.
<i>FM-STR_INDEX</i>	General for-loop index. Must be signed, at least 16 bits wide.
<i>FM-STR_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>FM-STR_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.

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