



MCUXpresso SDK Documentation

Release 26.03.00



NXP
Mar 24, 2026



Table of contents

1	FRDM-MCXA577	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	158
1.7	Middleware Documentation	158
1.7.1	MCU Boot	159
1.7.2	FreeMASTER	159
1.7.3	FreeRTOS	159
1.7.4	File systemFatfs	159
2	MCXA577	161
2.1	AOI: Crossbar AND/OR/INVERT Driver	161
2.2	CDOG	164
2.3	Clock Driver	168
2.4	CRC: Cyclic Redundancy Check Driver	212
2.5	CTIMER: Standard counter/timers	215
2.6	DAC: Digital-to-Analog Converter Driver	225
2.7	eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	230
2.8	eDMA core Driver	261
2.9	eDMA soc Driver	268
2.10	EIM: error injection module	269
2.11	ERM: error recording module	269
2.12	eSPI: Enhanced Serial Peripheral Interface	271
2.13	EWM: External Watchdog Monitor Driver	292
2.14	FGPIO Driver	294
2.15	FLASH Driver	294
2.16	FlexIO: FlexIO Driver	327
2.17	FlexIO Driver	327
2.18	FlexIO eDMA I2S Driver	344
2.19	FlexIO eDMA MCU Interface LCD Driver	348
2.20	FlexIO eDMA SPI Driver	350
2.21	FlexIO eDMA UART Driver	354
2.22	FlexIO I2C Master Driver	357
2.23	FlexIO I2S Driver	365
2.24	FlexIO MCU Interface LCD Driver	376
2.25	FlexIO SPI Driver	387
2.26	FlexIO UART Driver	400
2.27	FLEXSPI: Flexible Serial Peripheral Interface Driver	411
2.28	FLEXSPI eDMA Driver	427

2.29	FREQME: Frequency Measurement	430
2.30	GLIKEY	430
2.31	GLIKEY	430
2.32	GPIO: General-Purpose Input/Output Driver	434
2.33	GPIO Driver	437
2.34	I3C: I3C Driver	443
2.35	I3C Common Driver	445
2.36	I3C Master Driver	447
2.37	I3C Master DMA Driver	474
2.38	I3C Slave Driver	476
2.39	I3C Slave DMA Driver	489
2.40	INPUTMUX: Input Multiplexing Driver	492
2.41	KB Driver	562
2.42	Common Driver	564
2.43	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	578
2.44	Lpc_freqme	599
2.45	LPI2C: Low Power Inter-Integrated Circuit Driver	604
2.46	LPI2C Master Driver	606
2.47	LPI2C Master DMA Driver	620
2.48	LPI2C Slave Driver	622
2.49	LPSPi: Low Power Serial Peripheral Interface	633
2.50	LPSPi Peripheral driver	633
2.51	LPSPi eDMA Driver	654
2.52	LPTMR: Low-Power Timer	661
2.53	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	667
2.54	LPUART Driver	667
2.55	LPUART eDMA Driver	686
2.56	ENET: Ethernet Driver	689
2.57	Mcx_enet	689
2.58	MCX_SPC: System Power Control driver	717
2.59	NBOOT Driver	756
2.60	This type defines status return values used by NBOOT functions that are not easily	761
2.61	OSTIMER: OS Event Timer Driver	767
2.62	PORT: Port Control and Interrupts	771
2.63	Reset Driver	778
2.64	ROMAPI Driver	785
2.65	Runbootloader	785
2.66	RUNBOOTLOADER Driver	786
2.67	MCXN SMARTDMA Firmware	786
2.68	TENBASET_PHY: 10BASE-T1S Digital PHY Driver	789
2.69	TRDC: Trusted Resource Domain Controller	804
2.70	Trdc_core	826
2.71	Trdc_soc	840
2.72	TRNG: True Random Number Generator	840
2.73	UTICK: MicroTick Timer Driver	845
2.74	VREF: Voltage Reference Driver	846
2.75	WAKETIMER: WAKETIMER Driver	849
2.76	WUU: Wakeup Unit driver	852
2.77	WWDT: Windowed Watchdog Timer Driver	857
3	Middleware	861
3.1	Boot	861
3.1.1	MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource	861
3.1.2	MCUboot	862
3.2	File System	863
3.2.1	FatFs	863
3.3	Motor Control	865
3.3.1	FreeMASTER	865

4	RTOS	905
4.1	FreeRTOS	905
4.1.1	FreeRTOS kernel	905
4.1.2	FreeRTOS drivers	905
4.1.3	backoffalgorithm	905
4.1.4	corehttp	905
4.1.5	corejson	905
4.1.6	coremqtt	906
4.1.7	corepkcs11	906
4.1.8	freertos-plus-tcp	906

This documentation contains information specific to the frdm-mcxa577 board.

Chapter 1

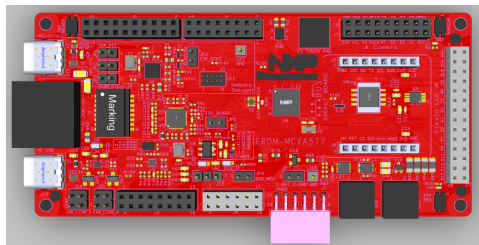
FRDM-MCXA577

1.1 Overview

FRDM-MCXA577 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 MCXA577 MCU, the FRDM-MCXA577 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: MCXA577
- PartNumber: MCXA577VPN

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.

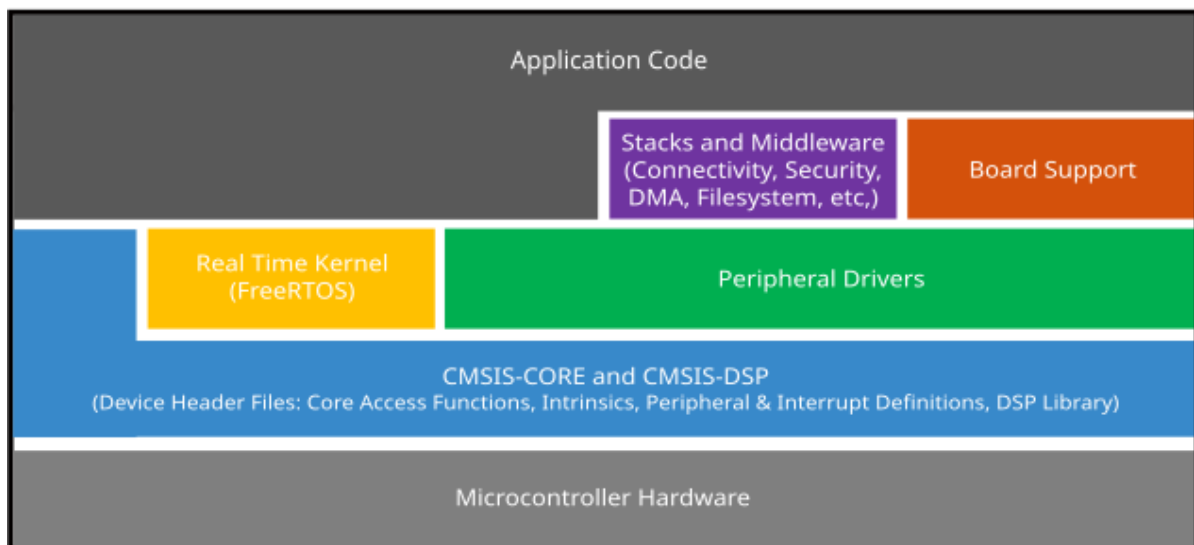
Note: The Repository-Layout SDK package was first introduced in version 25.09.00, but initially only for MCXW23x platforms.

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):

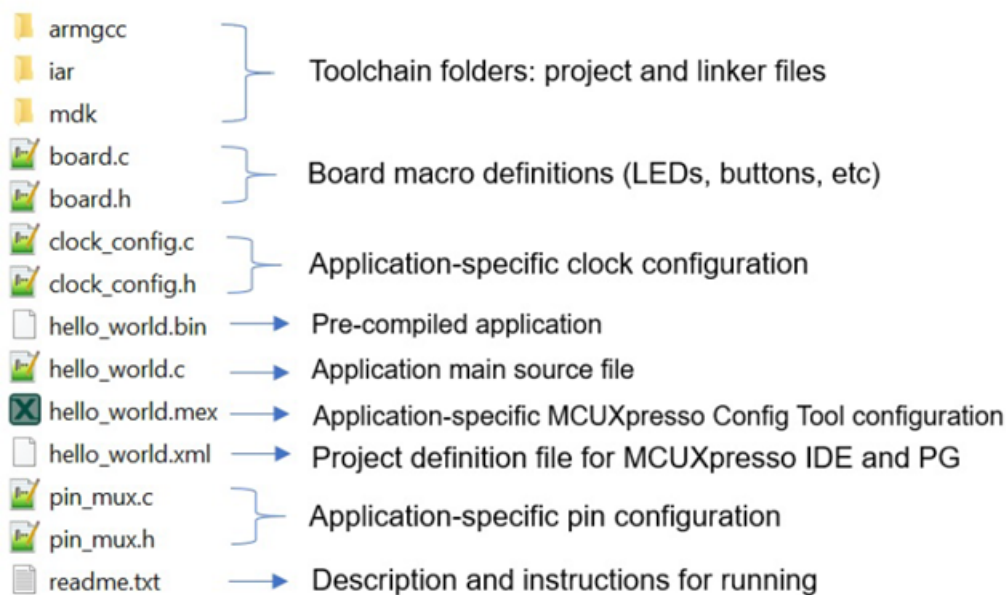
- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.

- **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.

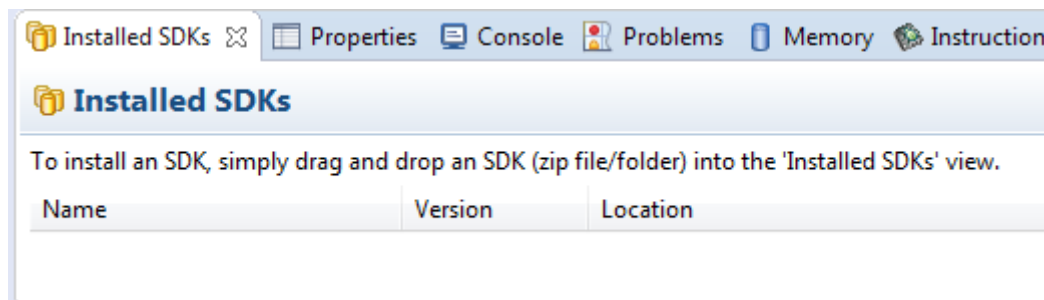
Run a demo using MCUXpresso IDE **Note:** Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

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.

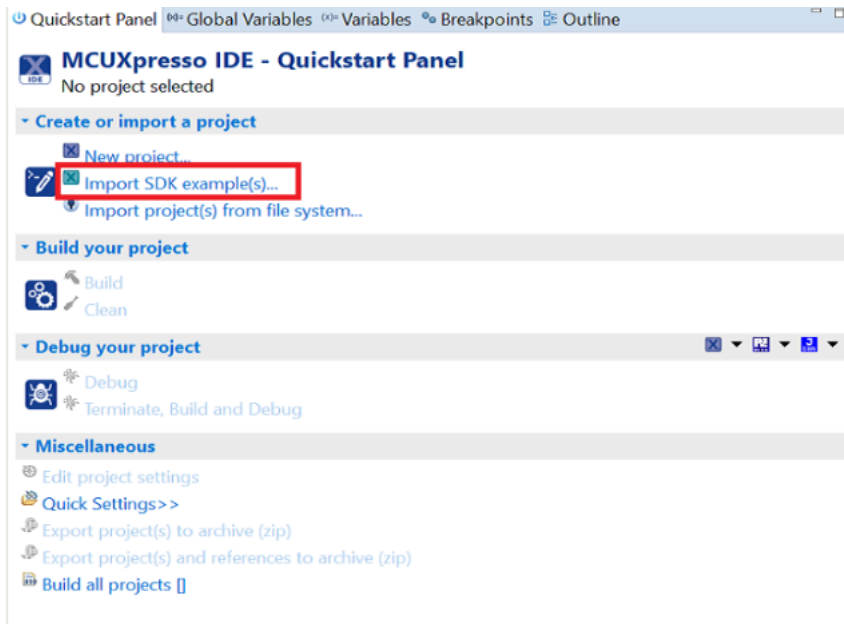
Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace.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

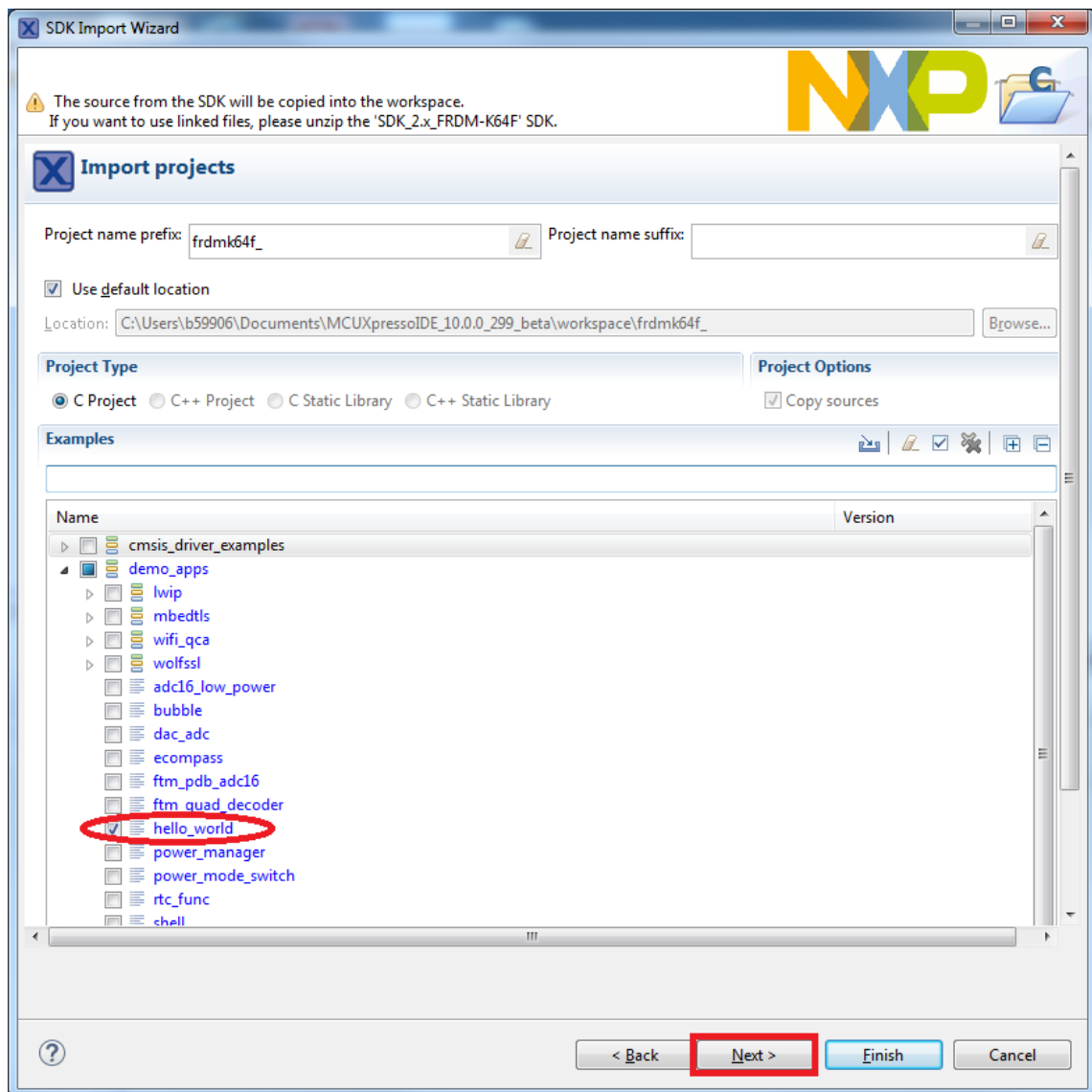
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



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



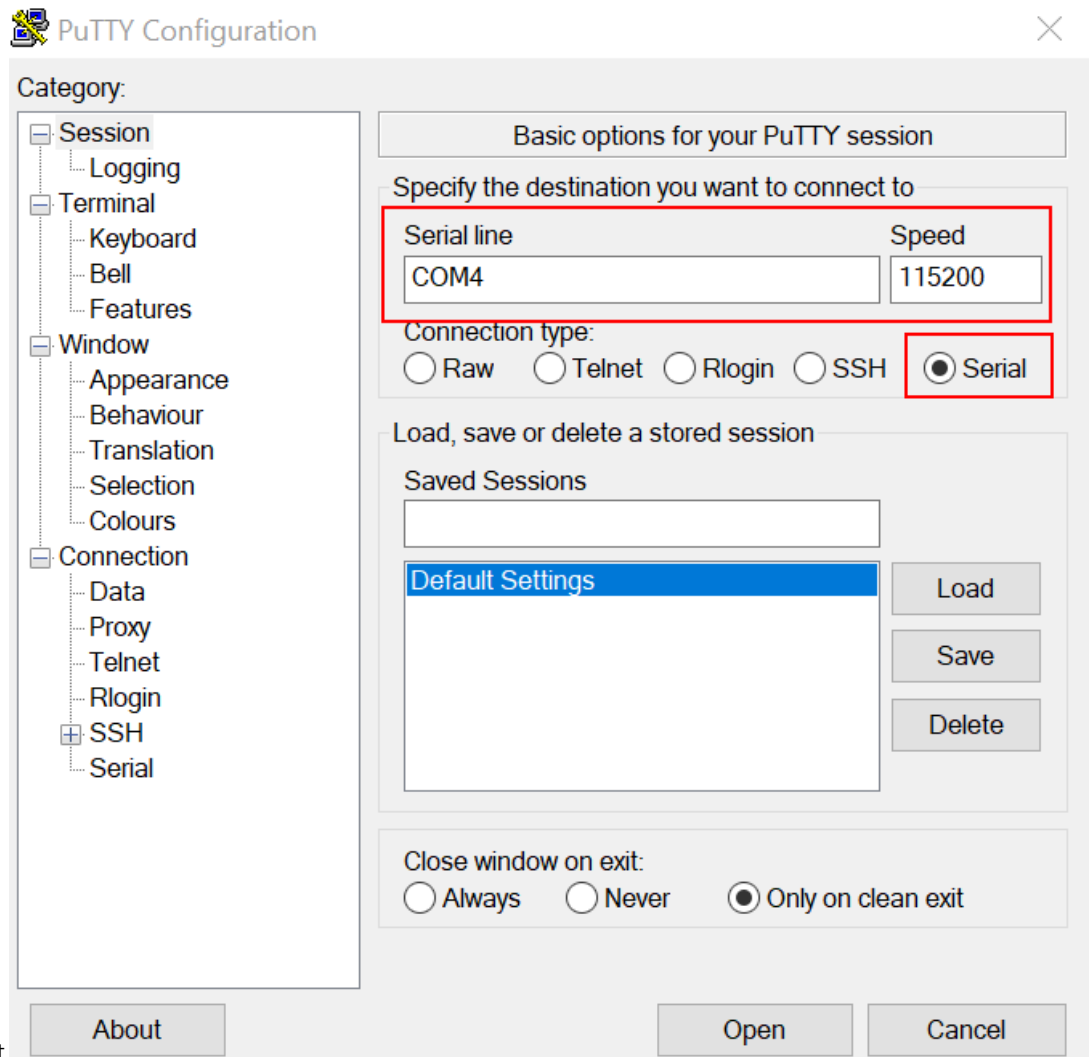
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

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:

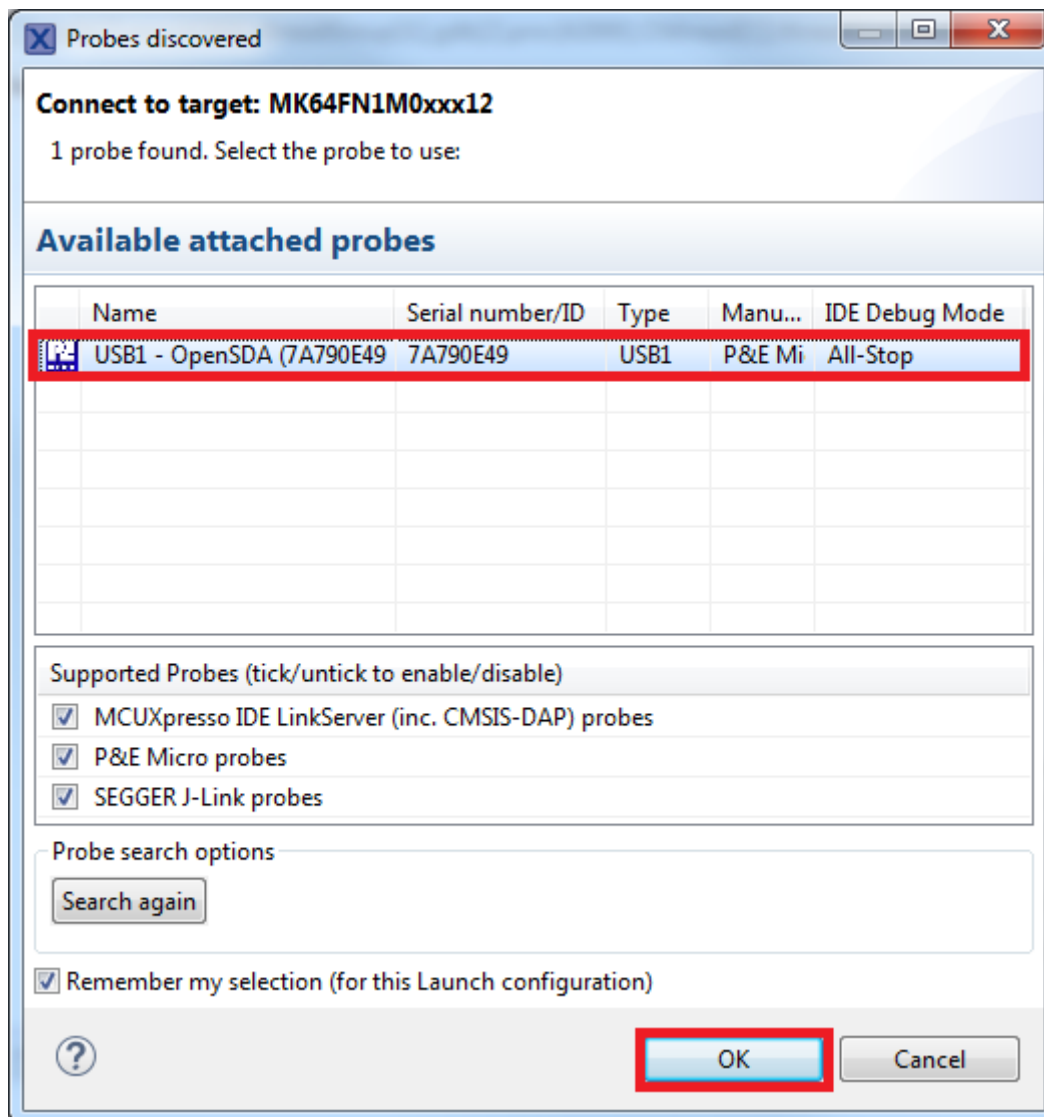
1. Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
2. Connect the development platform to your PC via a USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM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

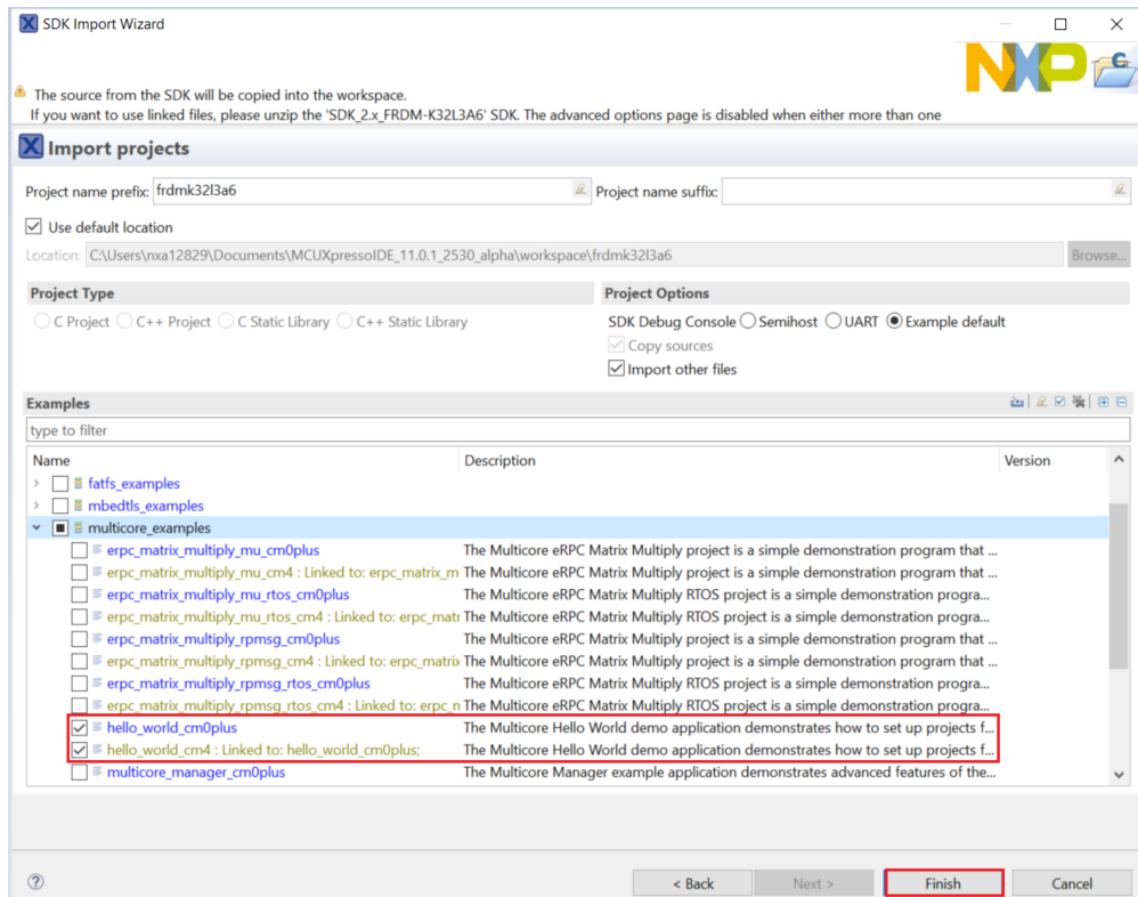


The `hello_world` application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.

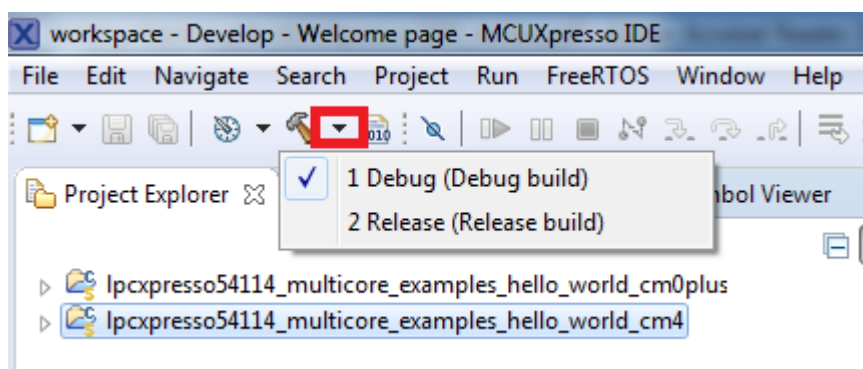


Build a multicore example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world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

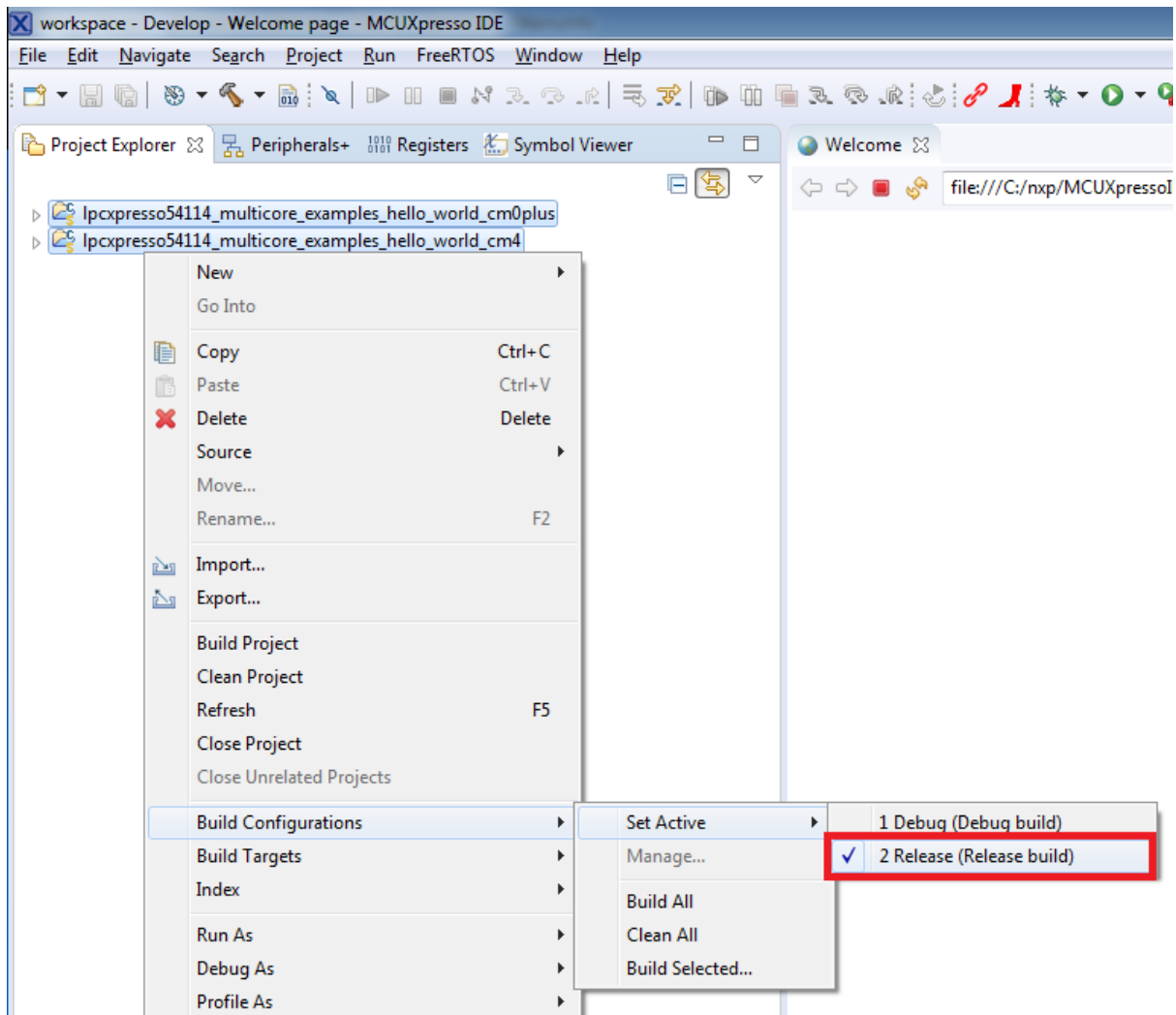


3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `lpcxpresso54114_multicore_examples_hello_world_cm4` project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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

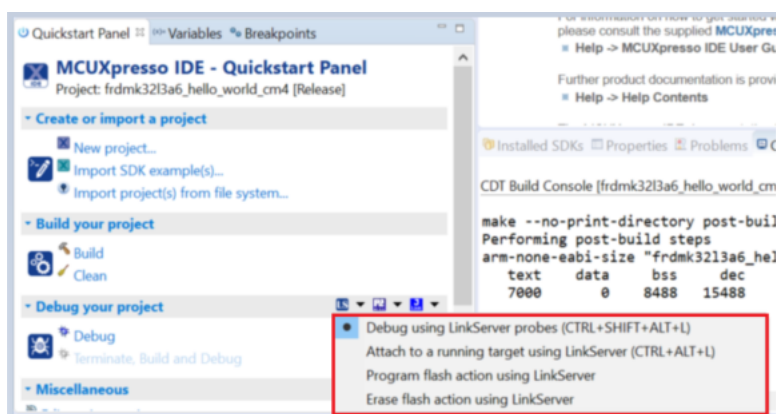


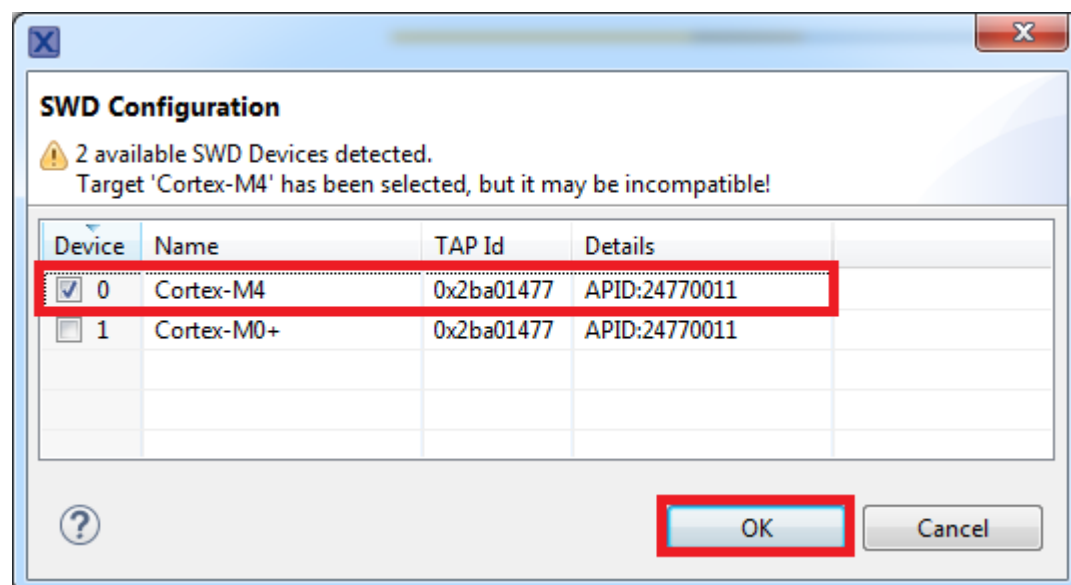
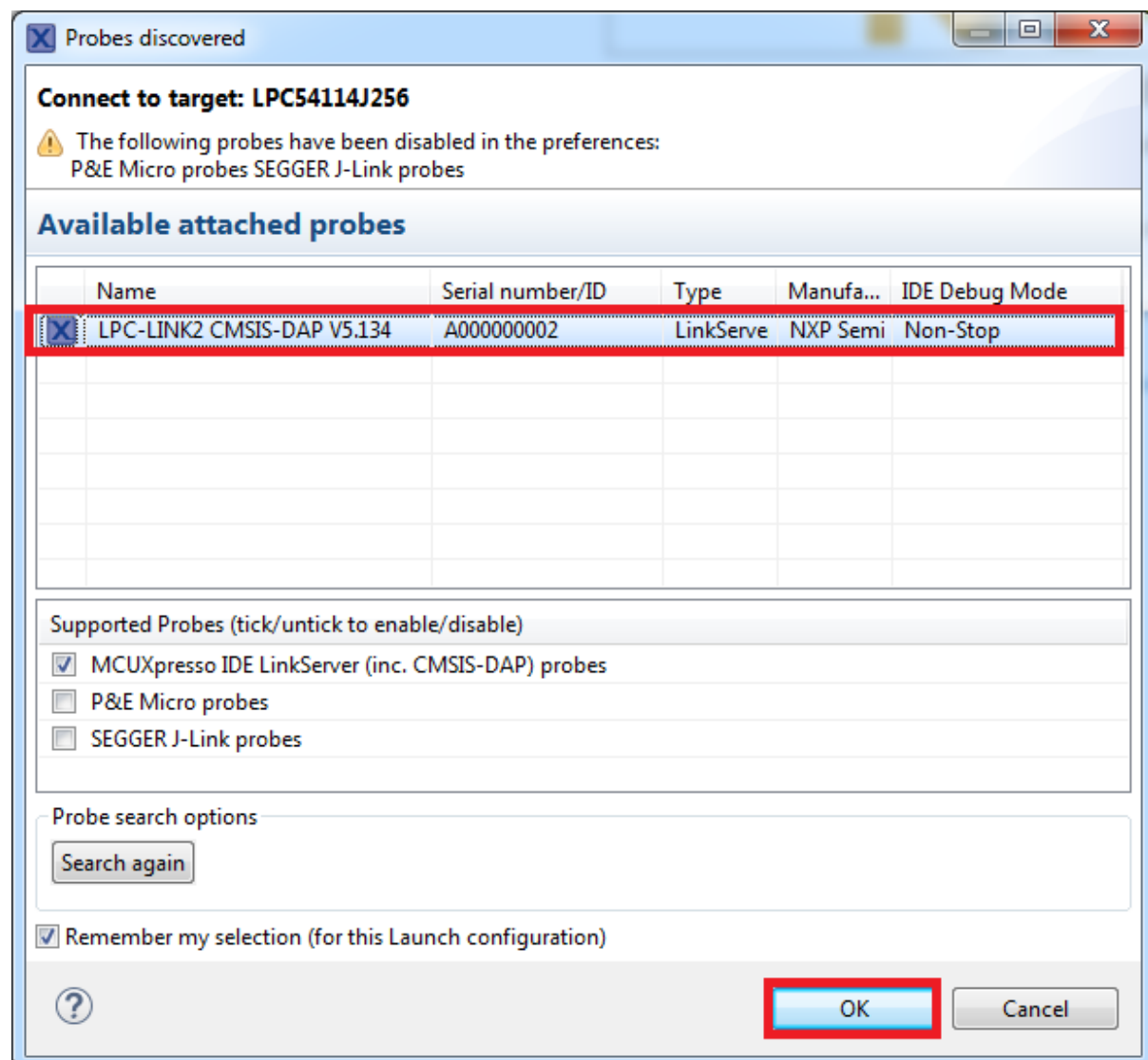
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

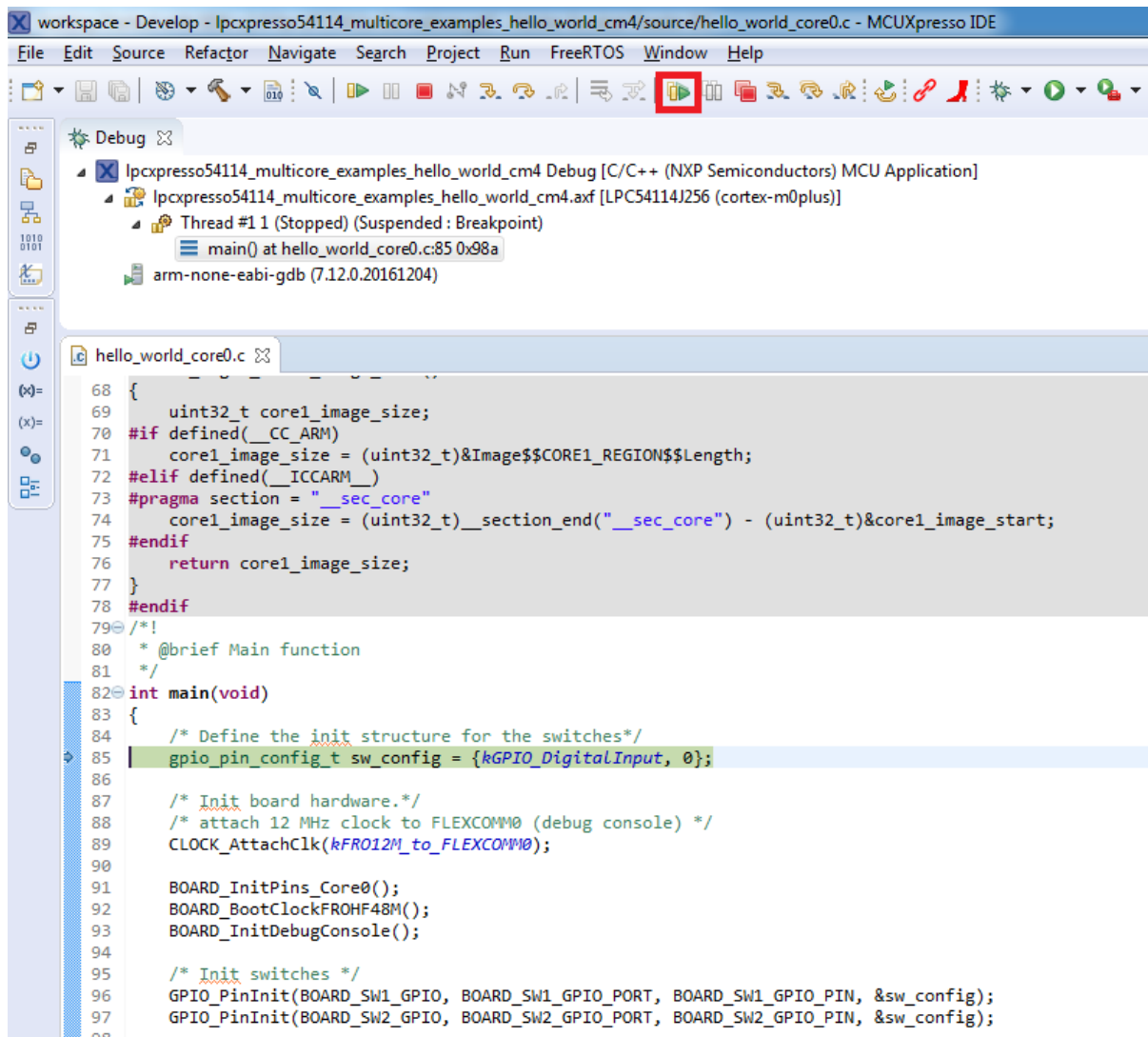
Note: When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build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



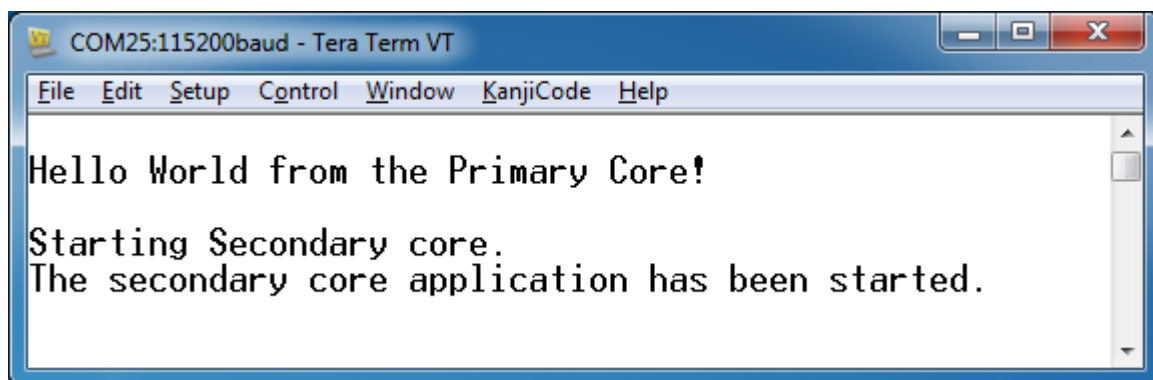
Run a multicore example application The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run an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





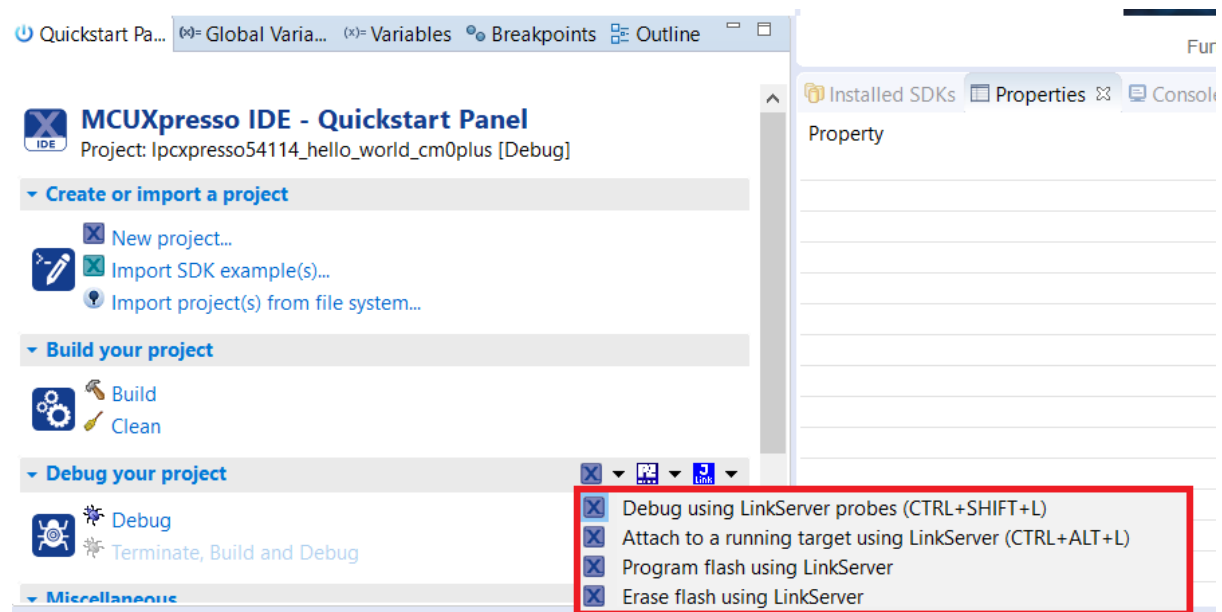


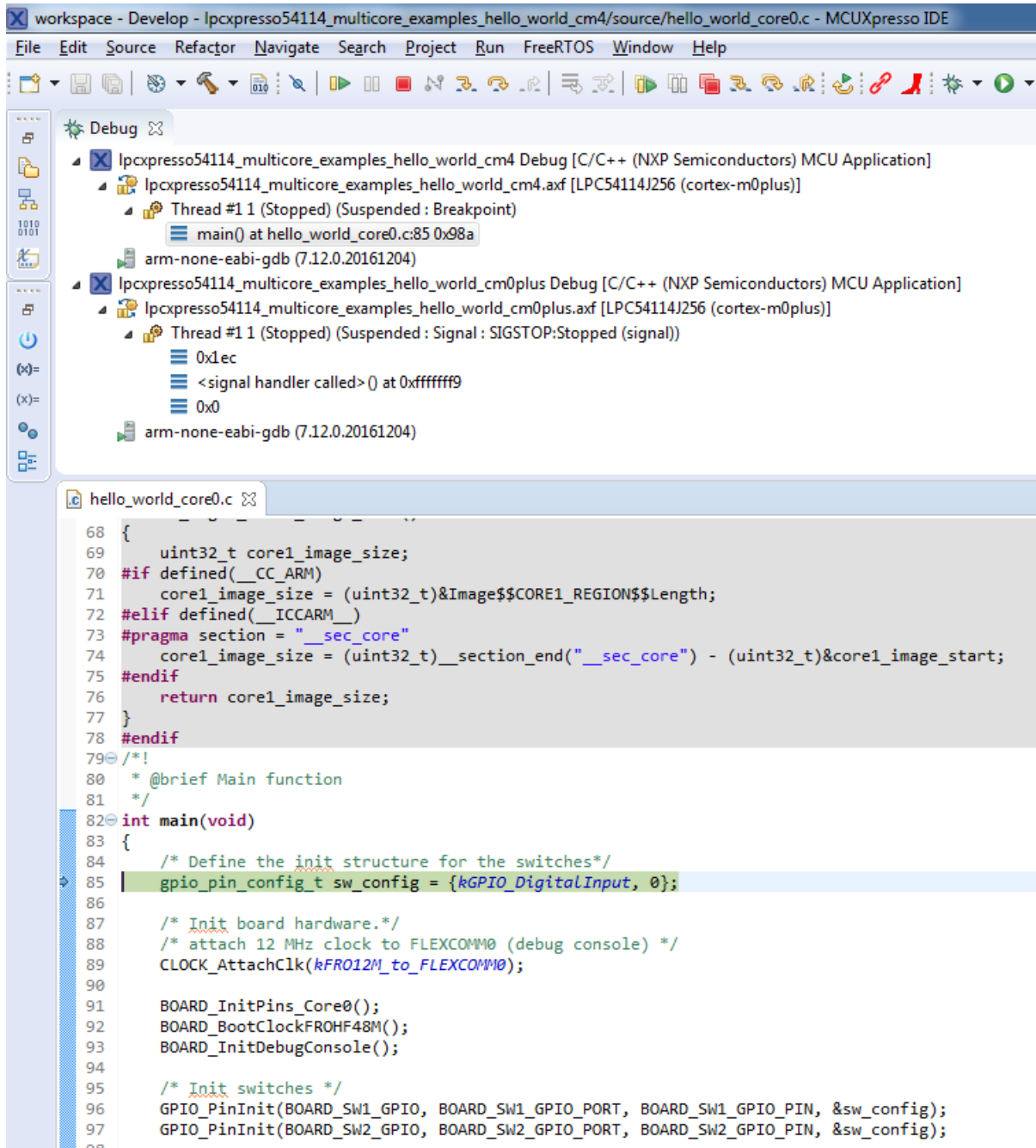
After clicking the “Resume All Debug sessions” button, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



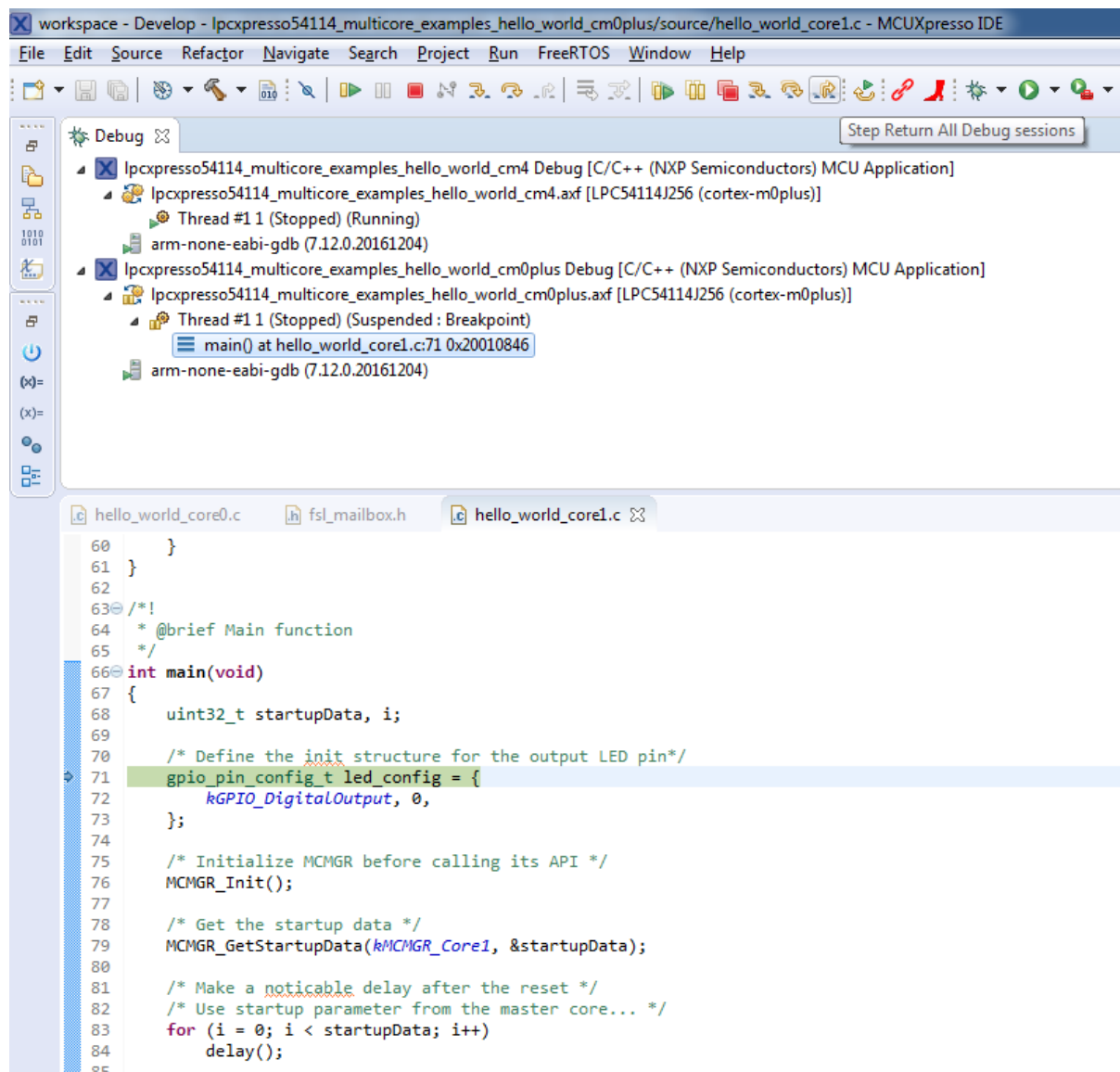
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps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

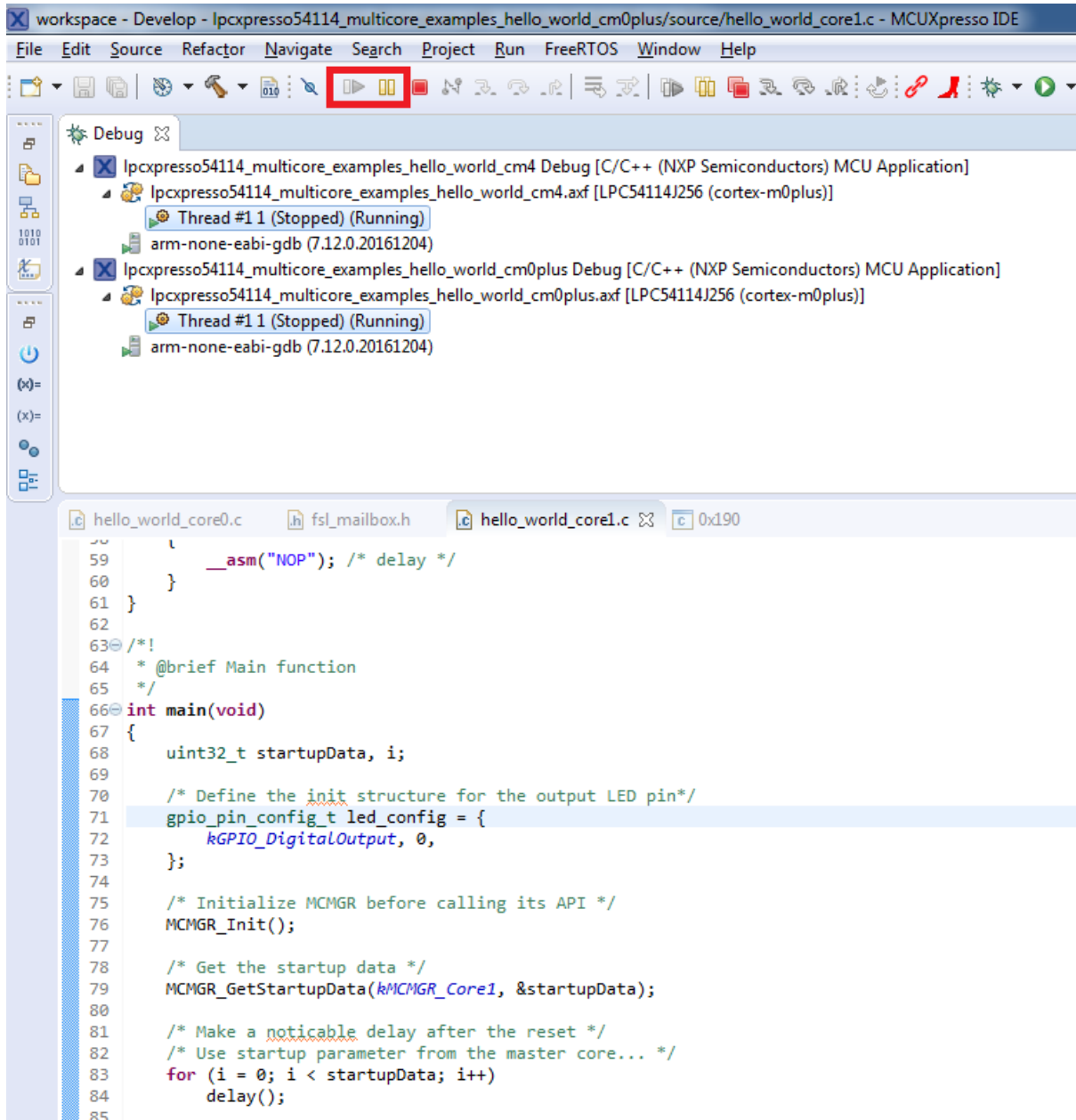


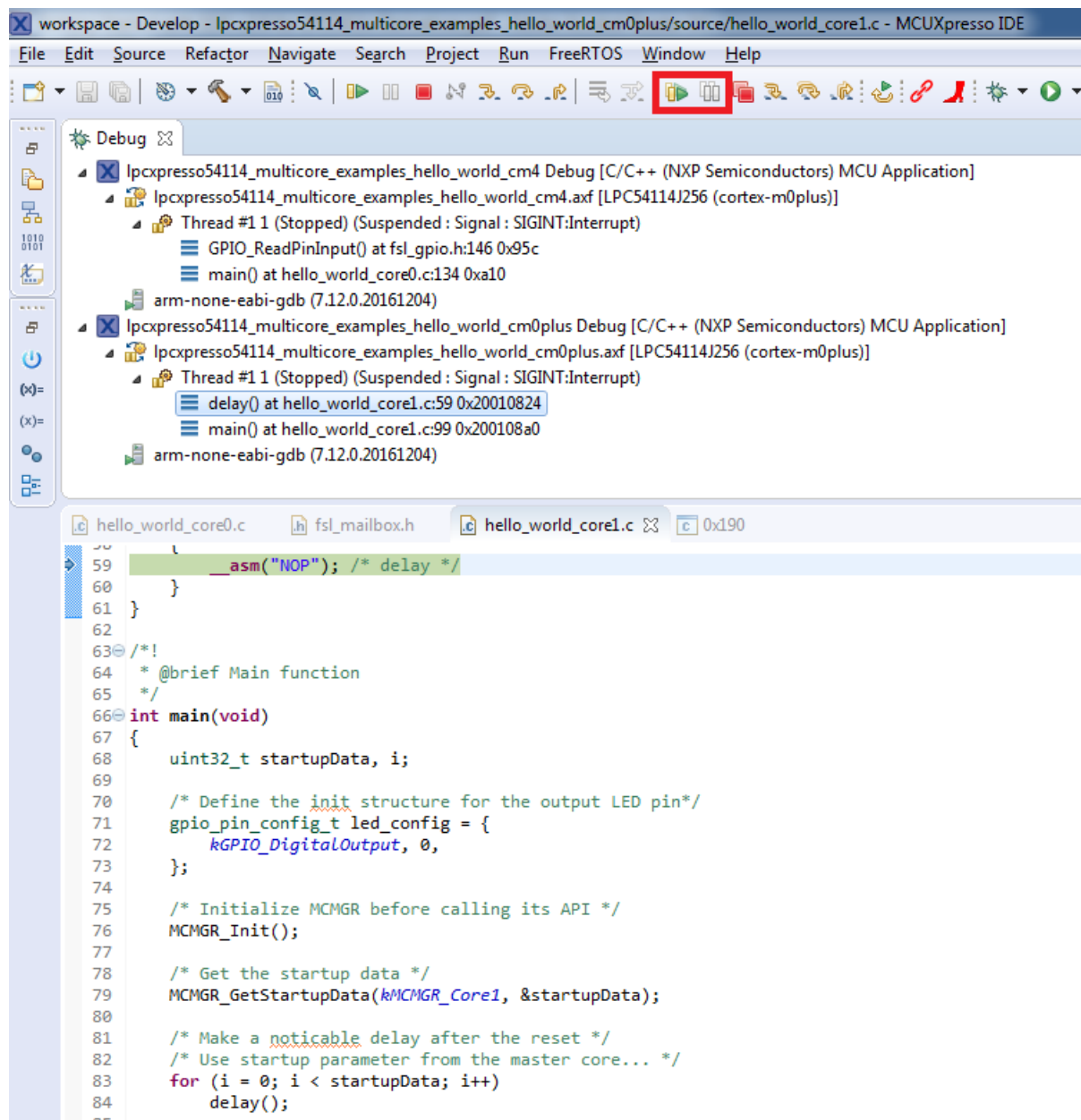


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello_world multicore application then starts running. The primary core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



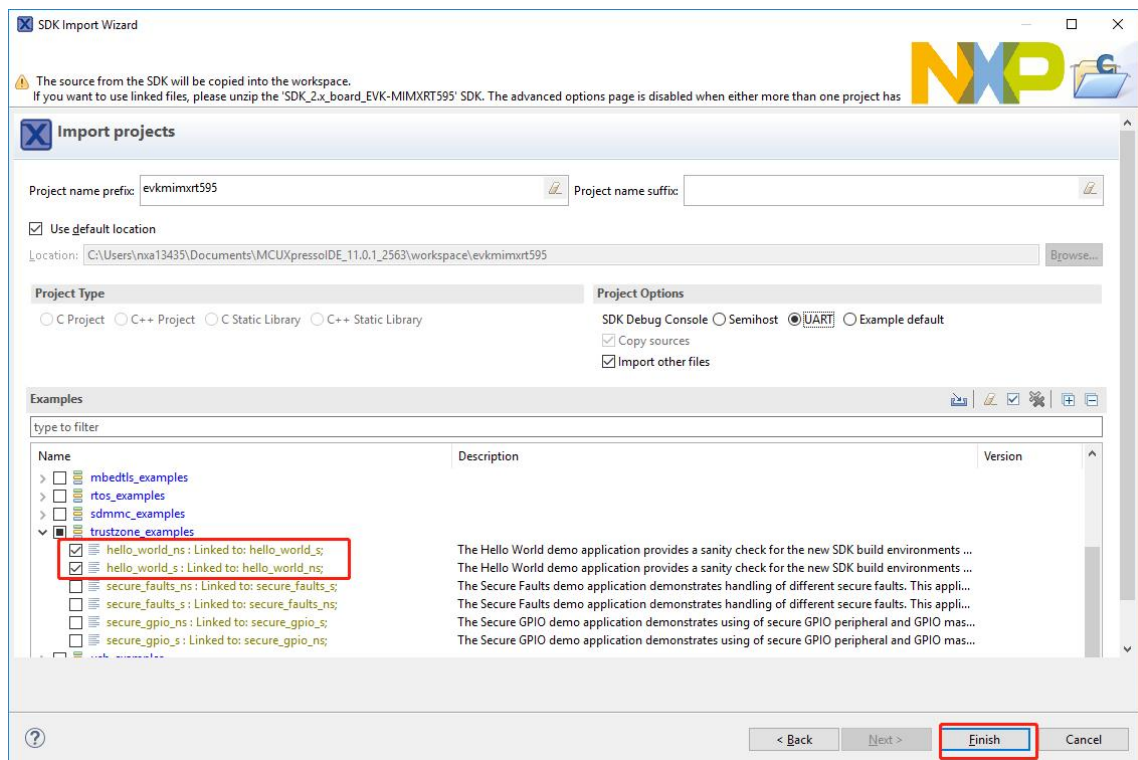
At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both the cores. This is done either by selecting both opened debug sessions (multiple selections) and clicking the “Suspend” / “Resume” control button,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



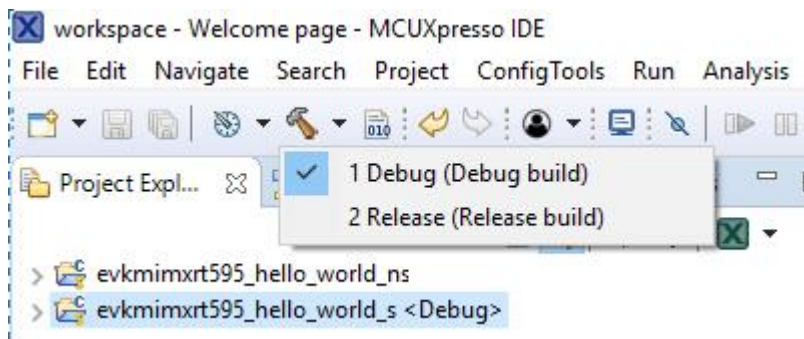


Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the `hello_world` example application targeted for the MIMXRT595-EVK hardware platform is used as an example, though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

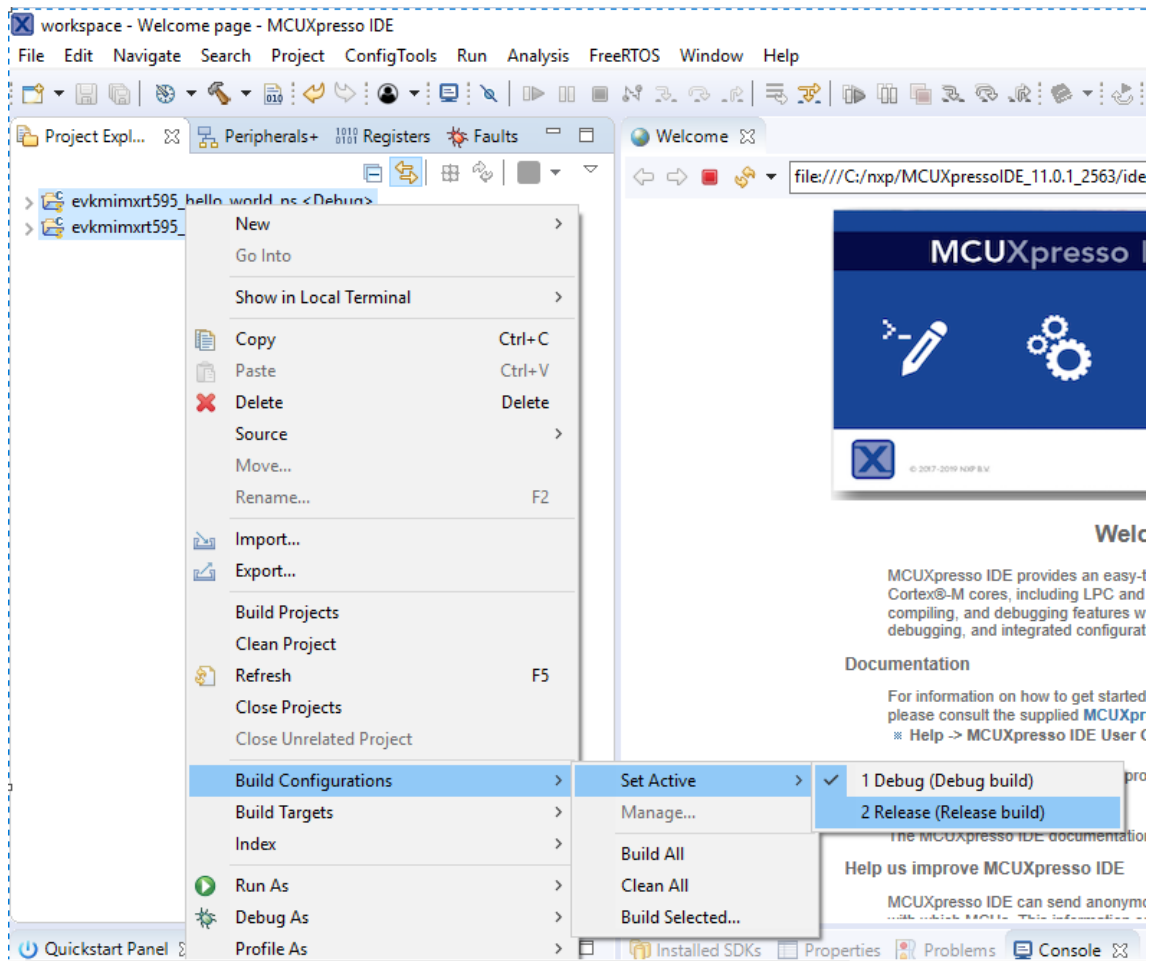


3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward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



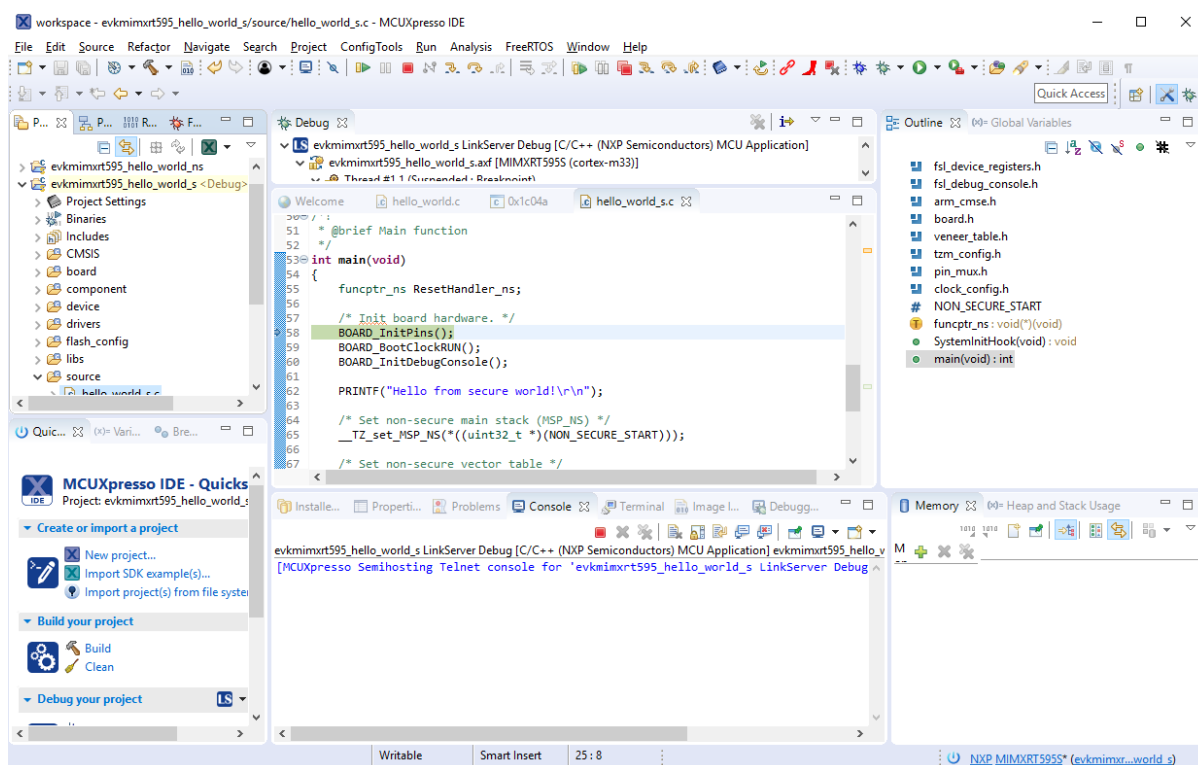
The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when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

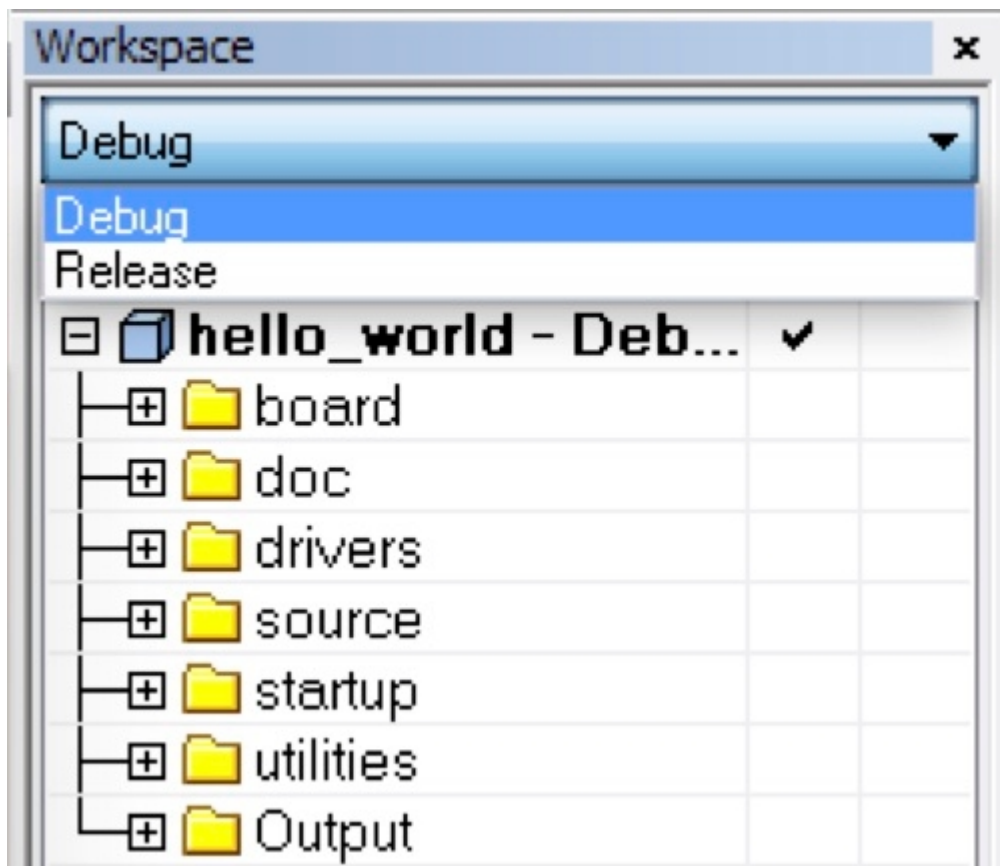
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

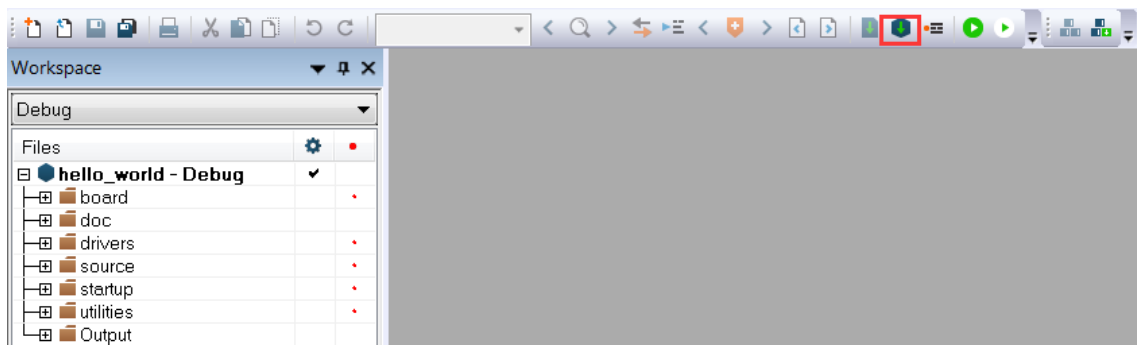
Other example applications may have additional folders in their path.

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

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



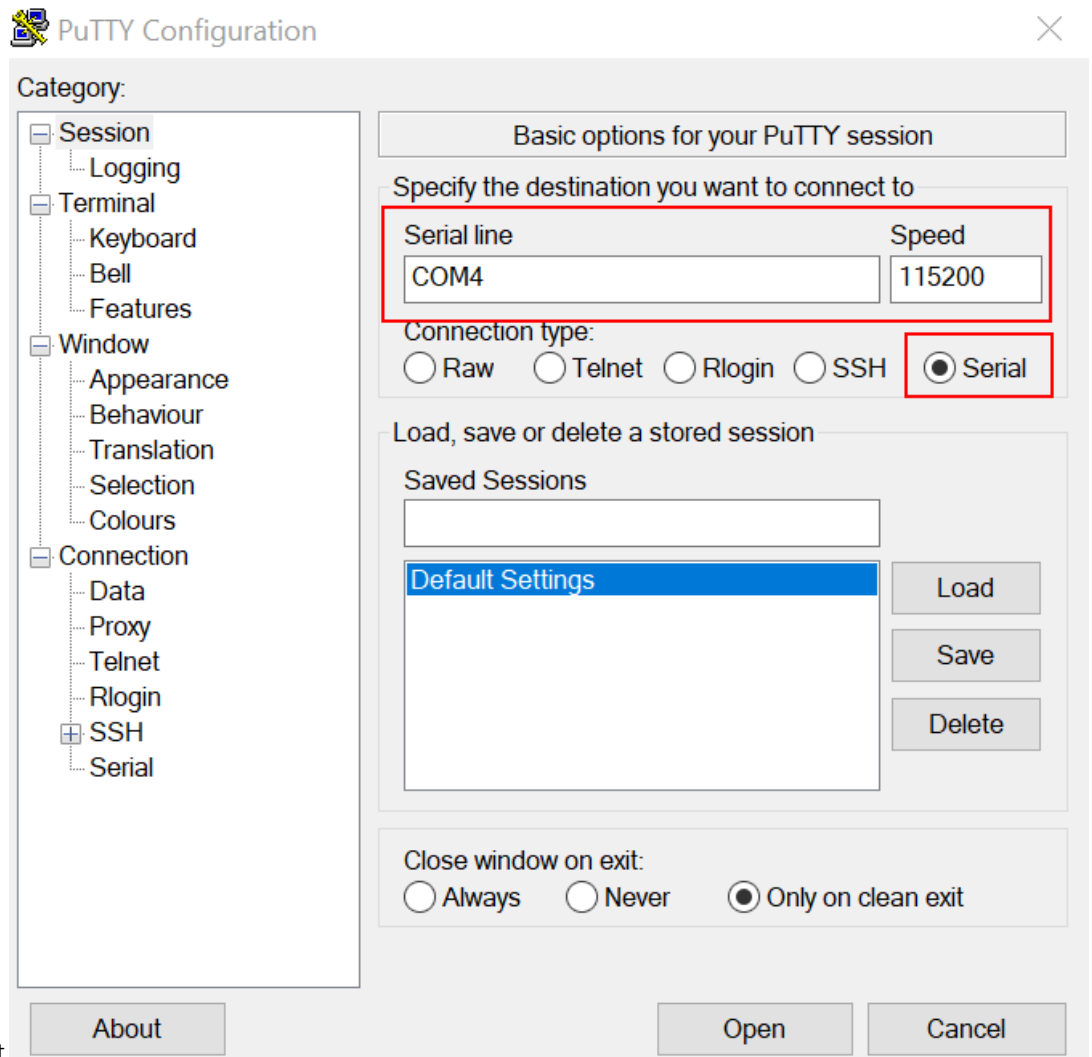
3. To build the demo application, click **Make**, highlighted in red in following figure.



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

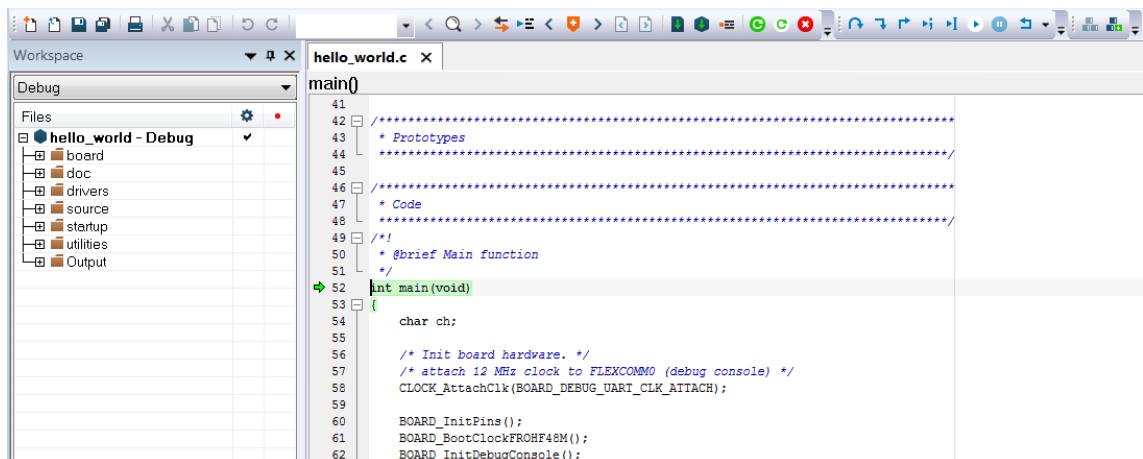


4. 1 stop bit

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
```

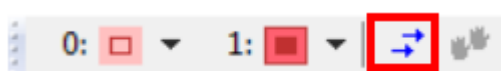
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

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.

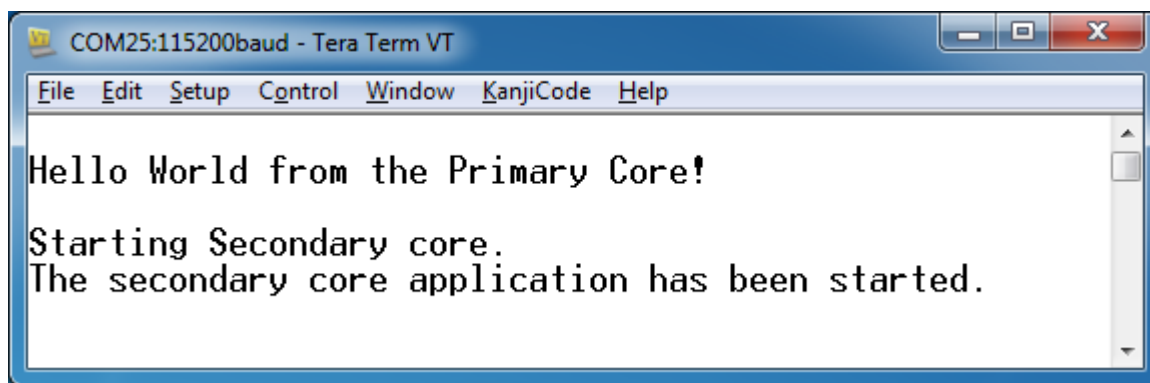
Run a multicore example application The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an example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

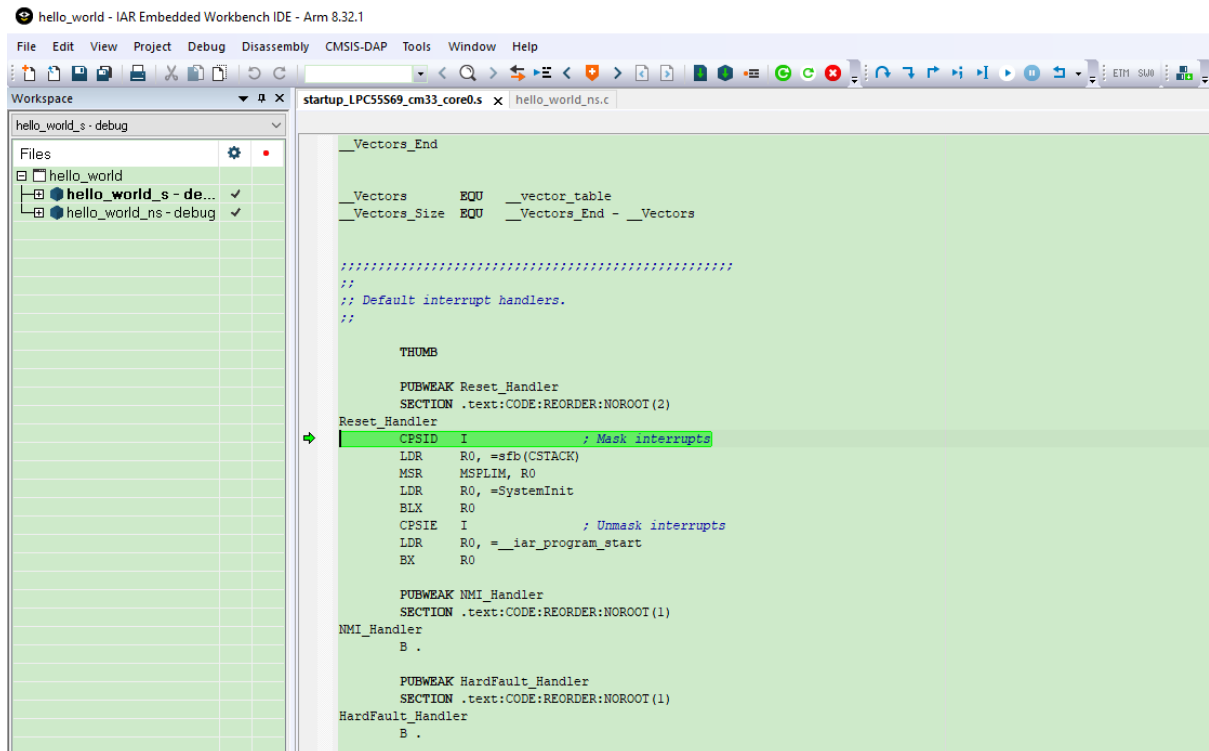
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_  
↪ns.eww
```

```
<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
```

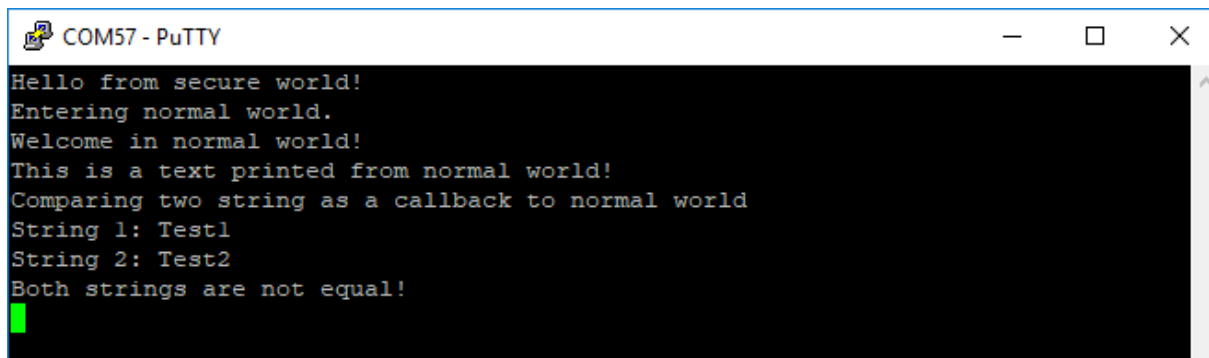
This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because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

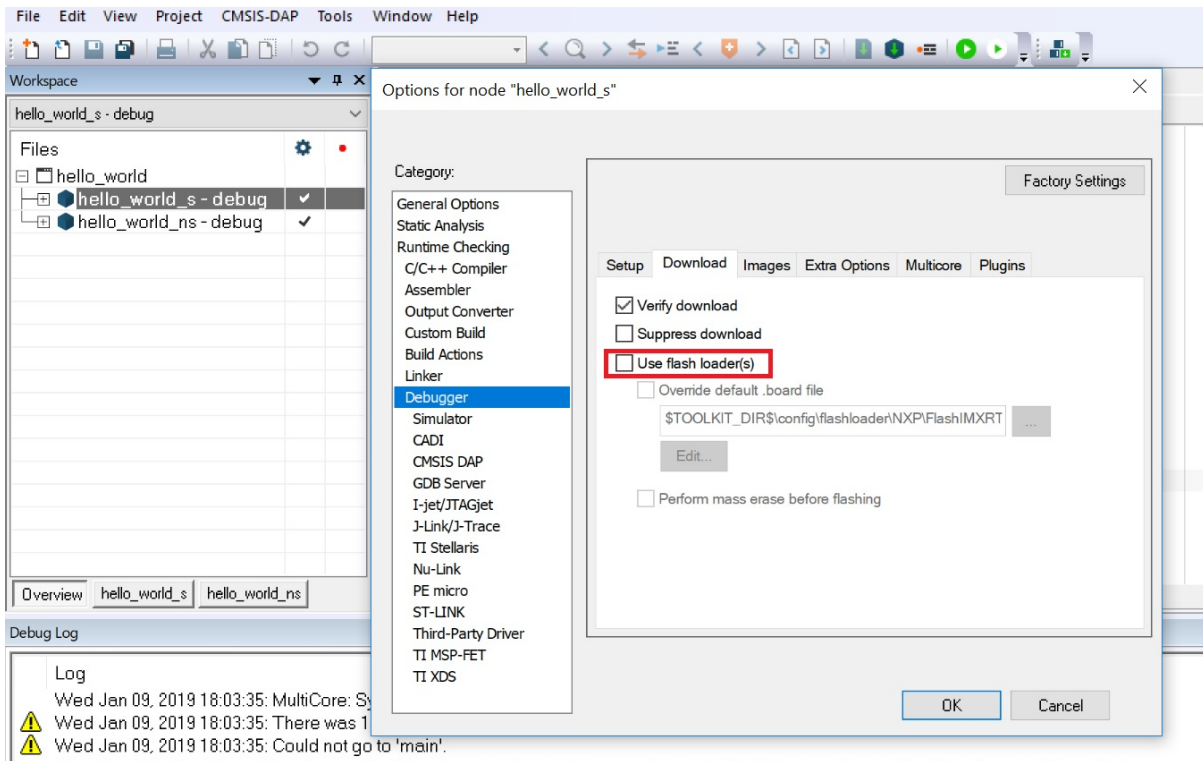


Run the code by clicking **Go** to start the application.

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.



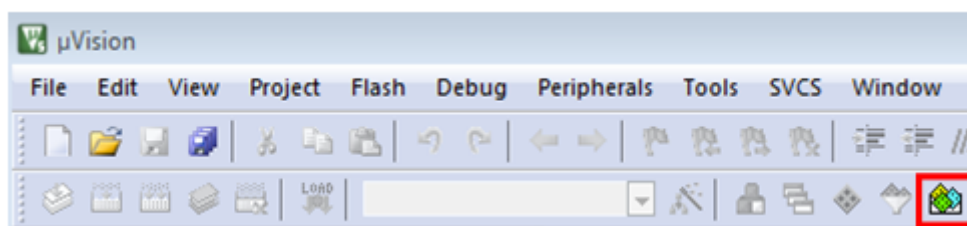
Note: If the application is running in RAM (debug/release build target), in **Options**>**Debugger > Download** tab, disable **Use flash loader(s)**. This can avoid the __ns download issue on i.MXRT500.



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.

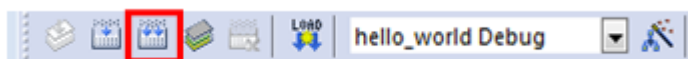
Build an example application

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:

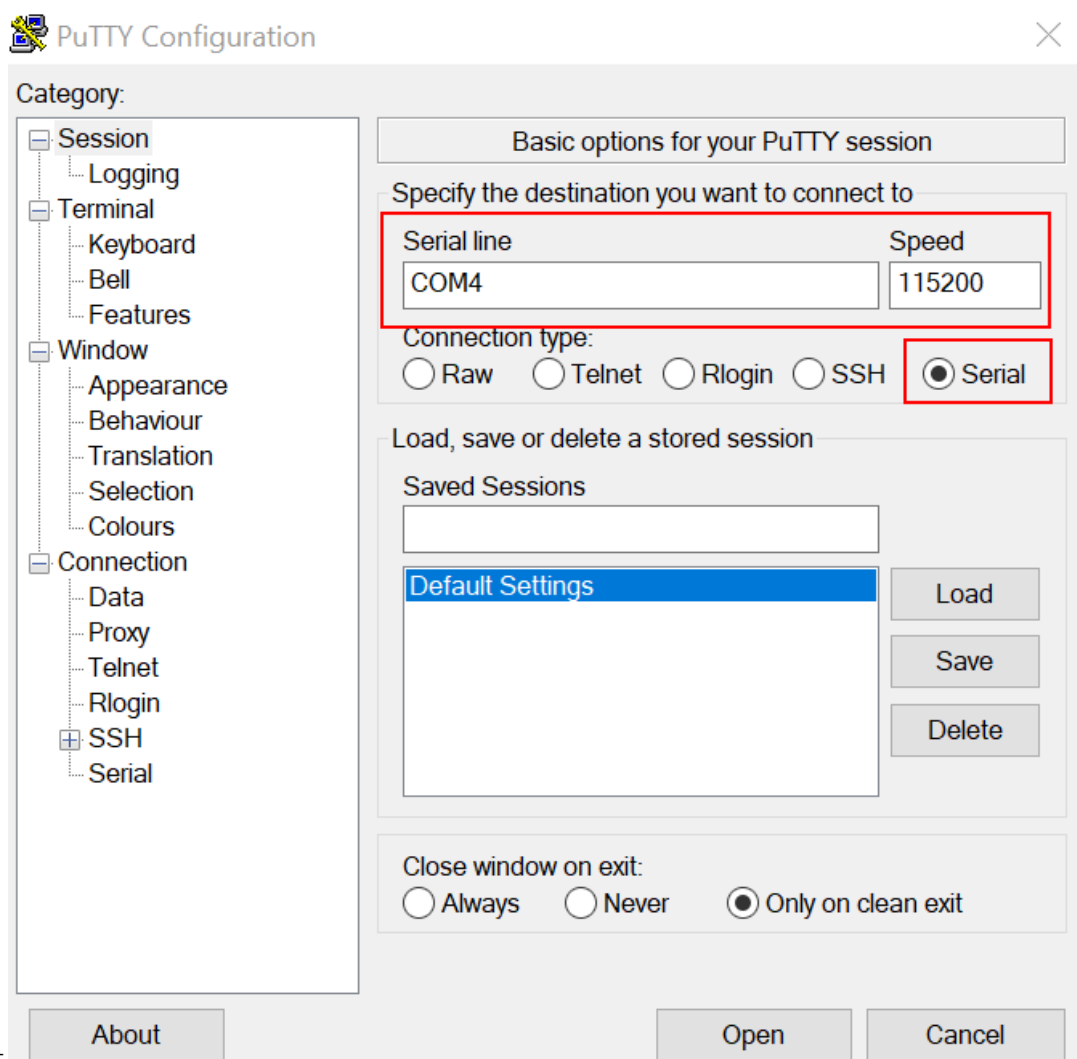
- To build the demo project, select **Rebuild**, highlighted in red.



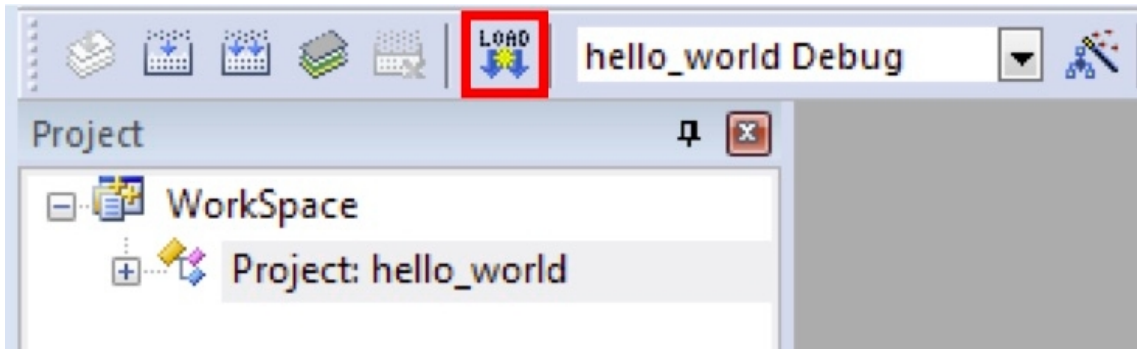
- The build completes without errors.

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

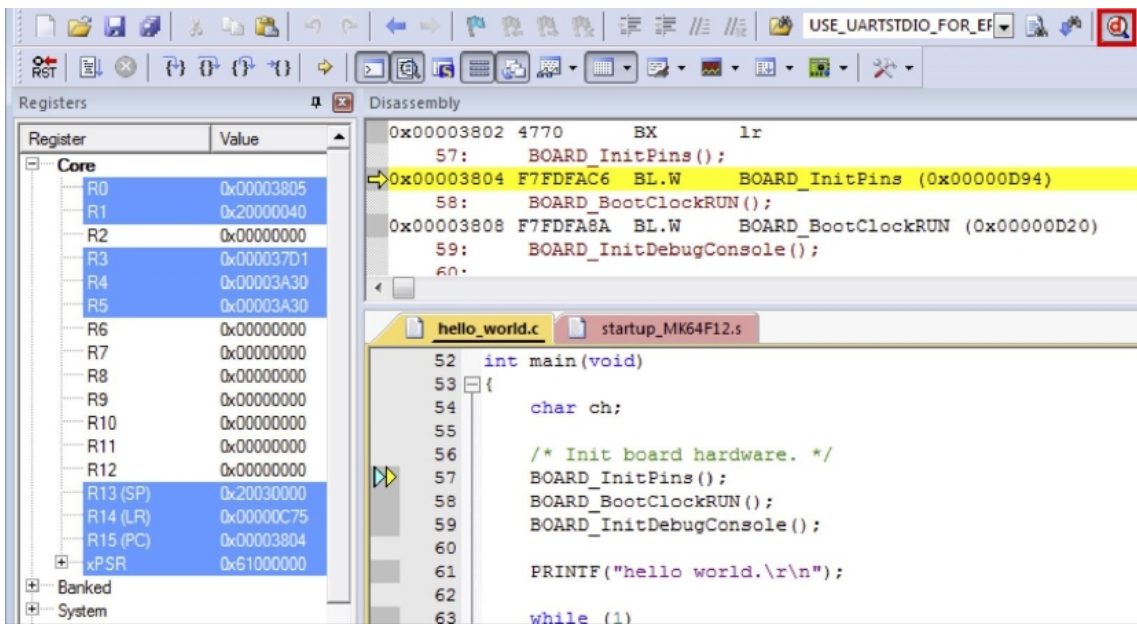
- Ensure the host driver for the debugger firmware has been installed. See [On-board debugger](#).
- Connect the development platform to your PC via USB cable using USB connector.
- Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine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



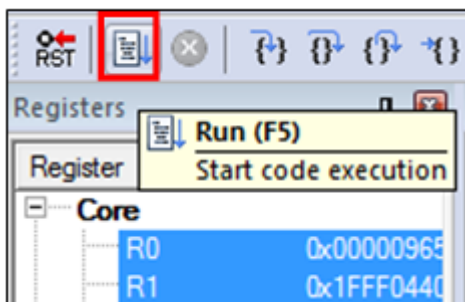
- 1 stop bit
- In μ Vision, after the application is built, click the **Download** button to download the application to the target.



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



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



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



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

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

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

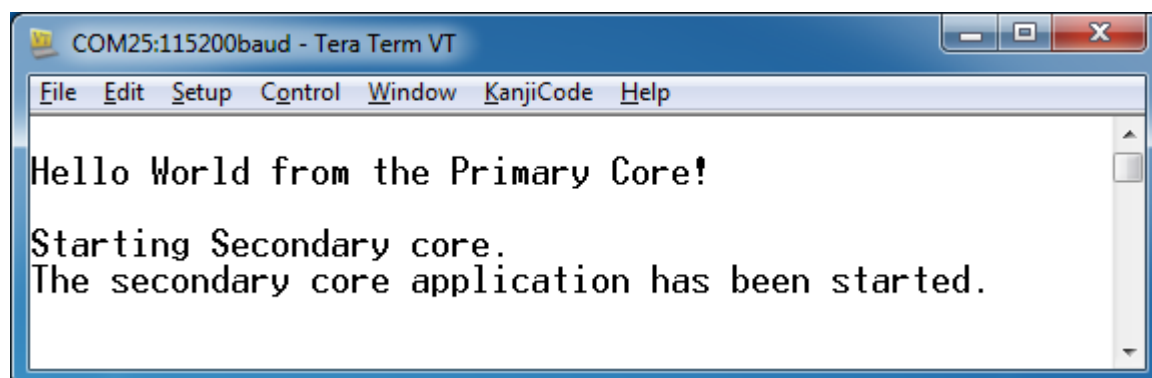
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
→cm0plus.uvmpw
```

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

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

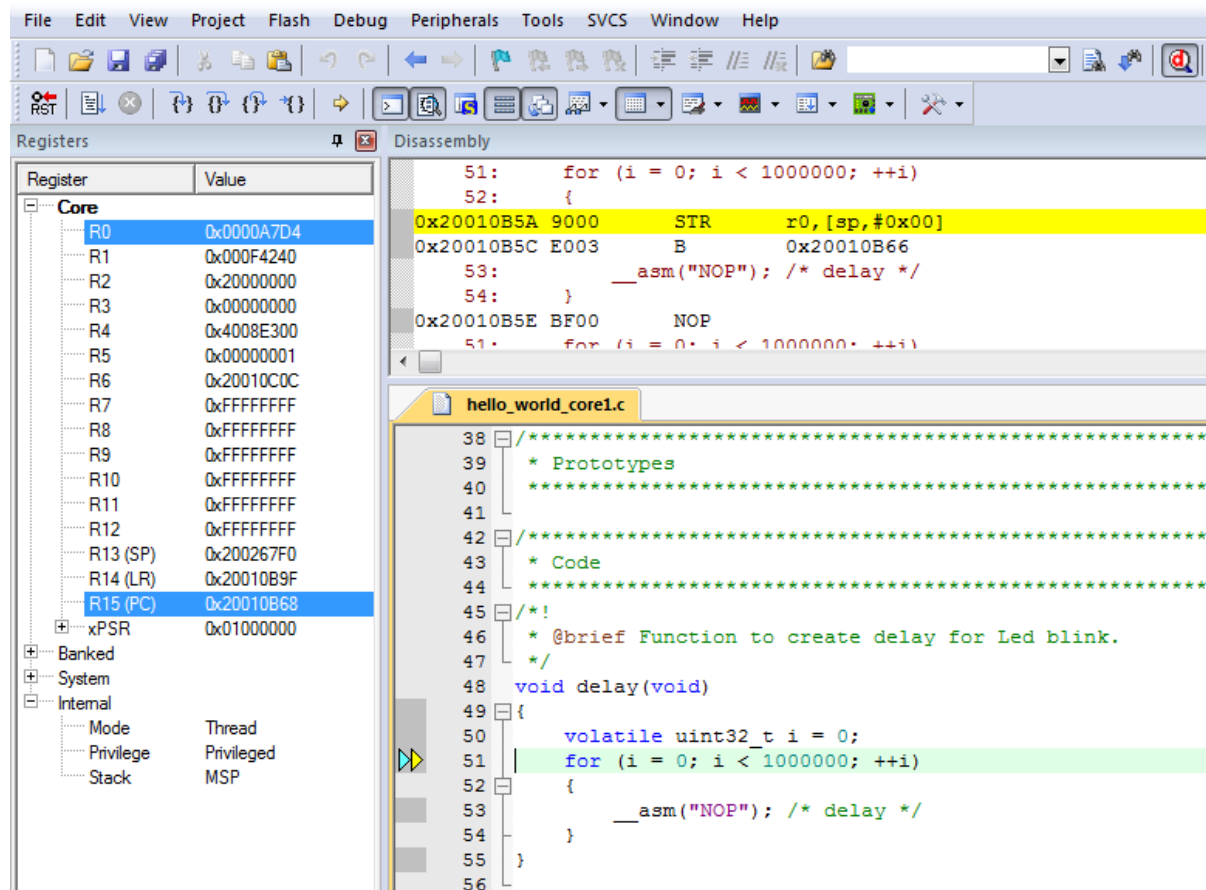
Run a multicore example application The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

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



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

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



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

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

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

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

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

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

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

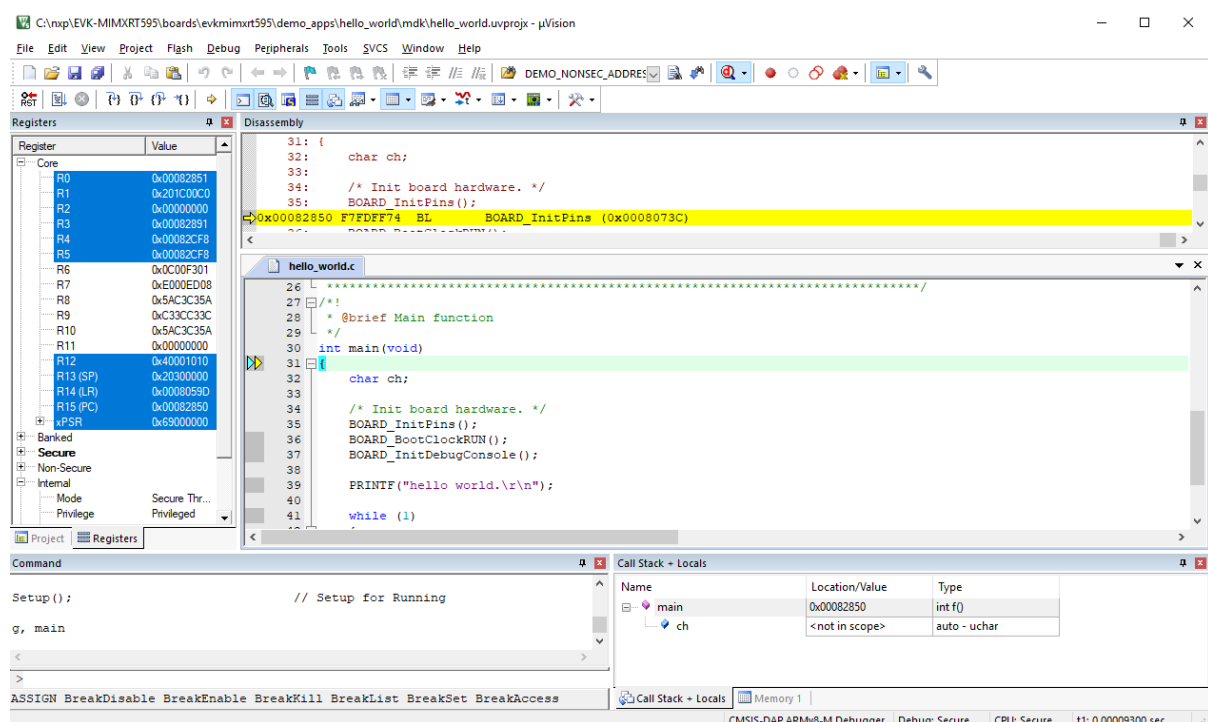
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.  
→ uvmpw
```

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

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

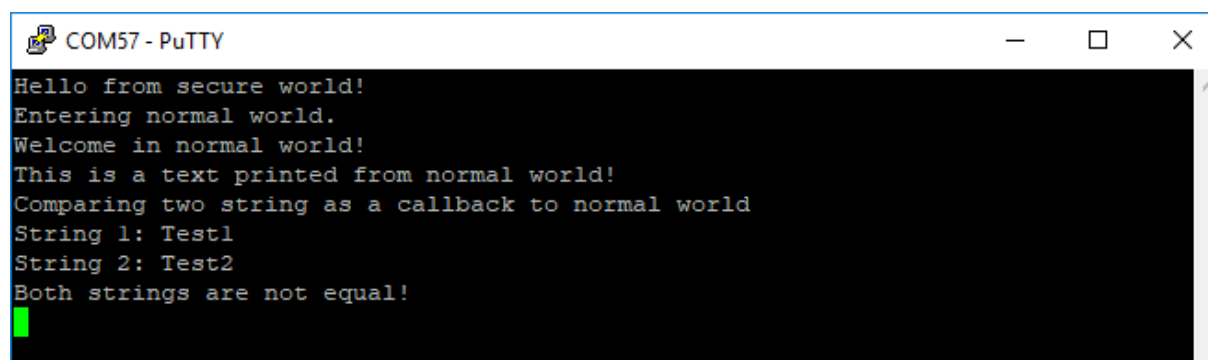
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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



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

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



Run a demo using 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.

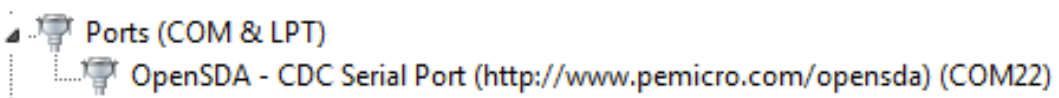
2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

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.

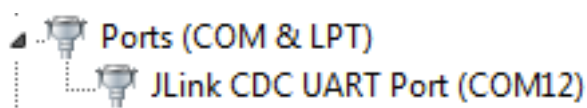
1. **CMSIS-DAP/mbed/DAPLink** interface:



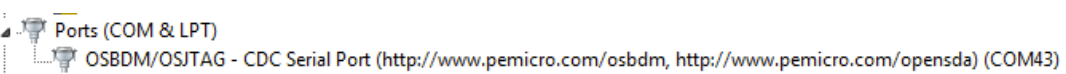
2. **P&E Micro:**



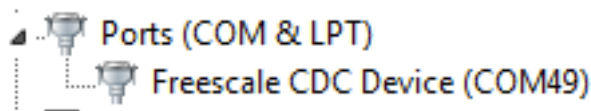
3. **J-Link:**



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.

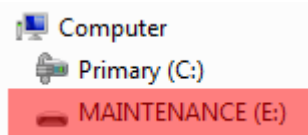
- For boards with CMSIS-DAP firmware, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- For boards with a P&E Micro interface, see [PE micro](#) to download 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-MIMX8MND3L	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-MCXN236	CMSIS-DAP	MCU-Link
FRDM-MCXN947	CMSIS-DAP	MCU-Link
FRDM-MCXW23	CMSIS-DAP	MCU-Link
FRDM-MCXW71	CMSIS-DAP	MCU-Link
FRDM-MCXW72	CMSIS-DAP	MCU-Link
FRDM-RW612	CMSIS-DAP	MCU-Link
IMX943-EVK	N/A	N/A
IMX95LP4XEVK-15	N/A	N/A
IMX95LPD5EVK-19	N/A	N/A
IMX95VERDINEVK	N/A	N/A
KW45B41Z-EVK	CMSIS-DAP	MCU-Link
KW45B41Z-LOC	CMSIS-DAP	MCU-Link
KW47-EVK	CMSIS-DAP	MCU-Link
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

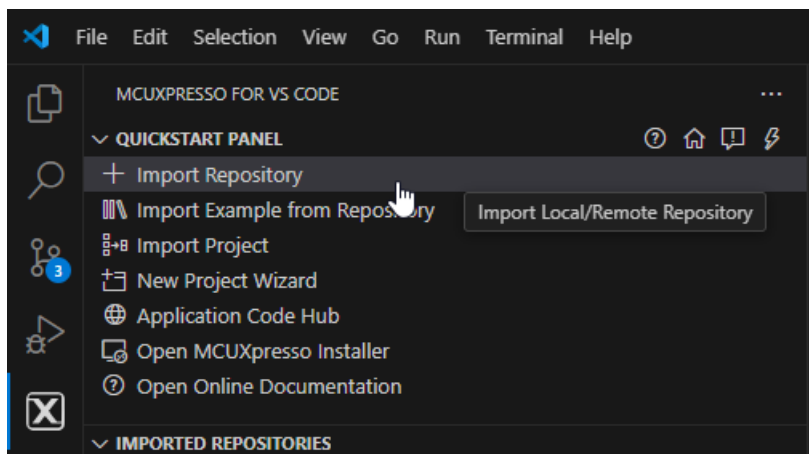
Extension Installation

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

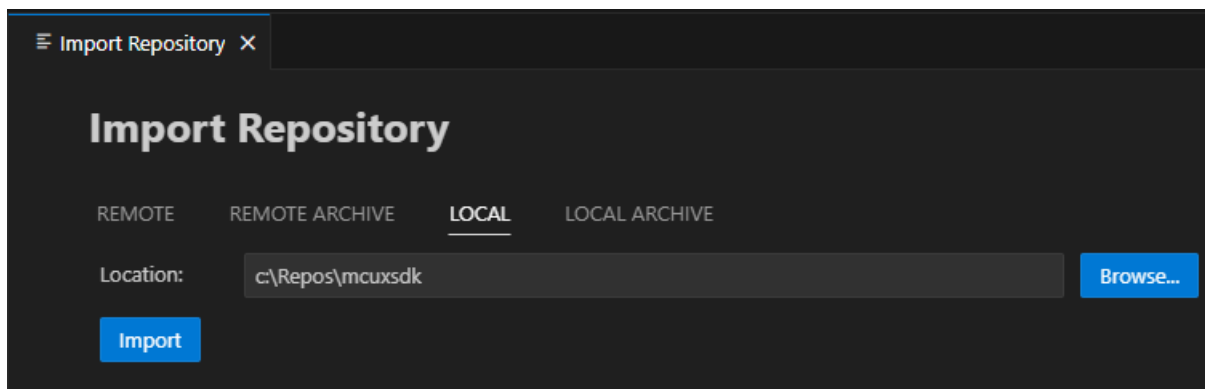
Import Methods The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

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

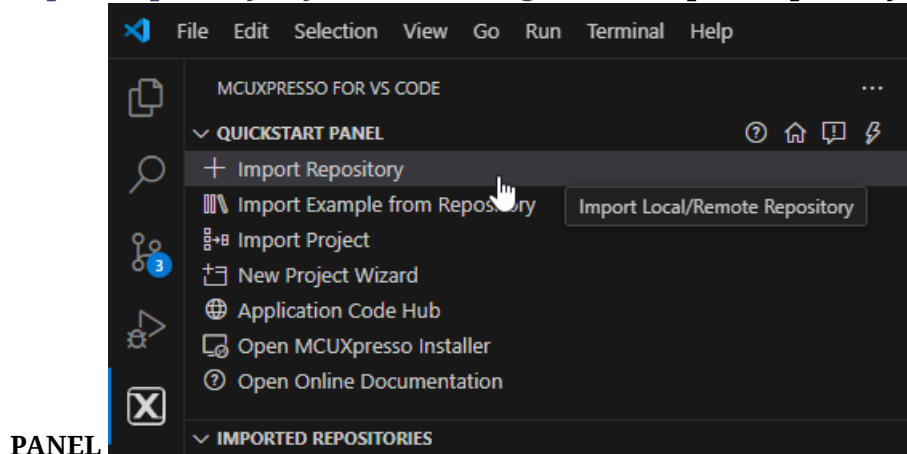


Note: You can import the SDK in several ways. Refer to the [MCUXpresso for VS Code documentation](#) for details.

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

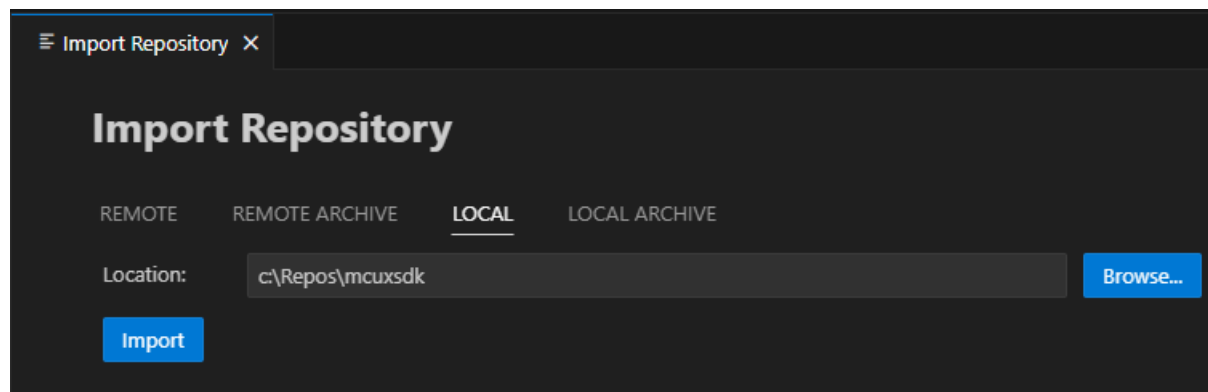


Import Repository-Layout SDK Package Click **Import Repository** from the **QUICKSTART**

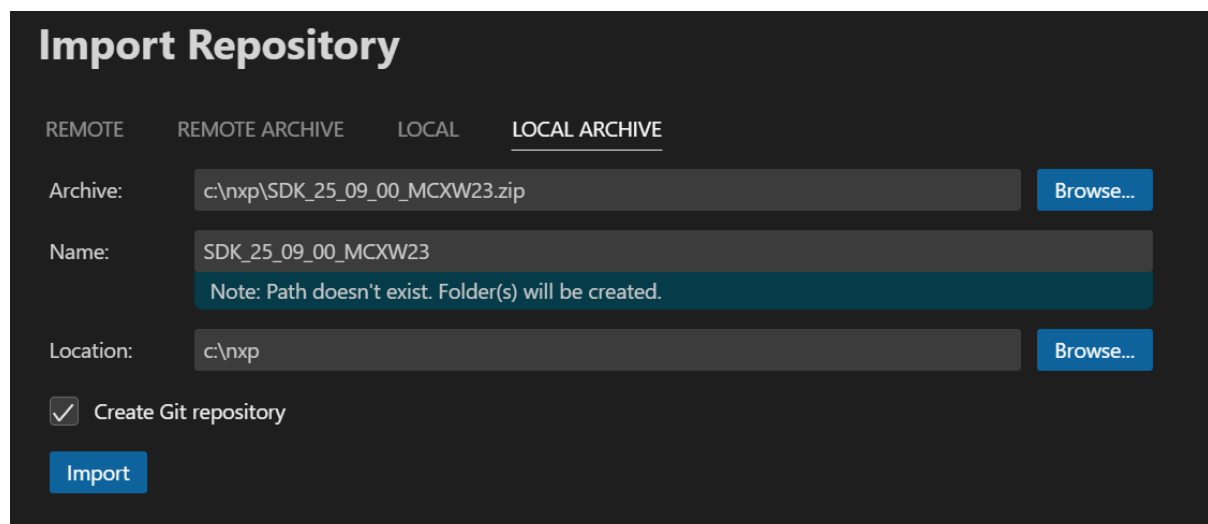


PANEL

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



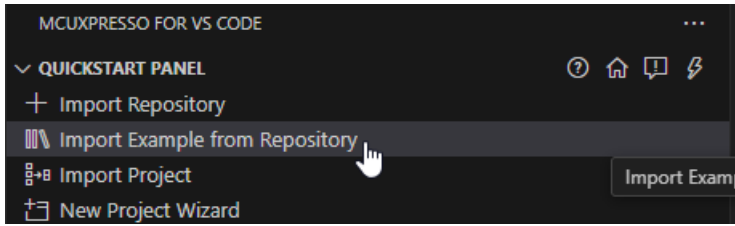
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



Building Example Applications

Import Example Project

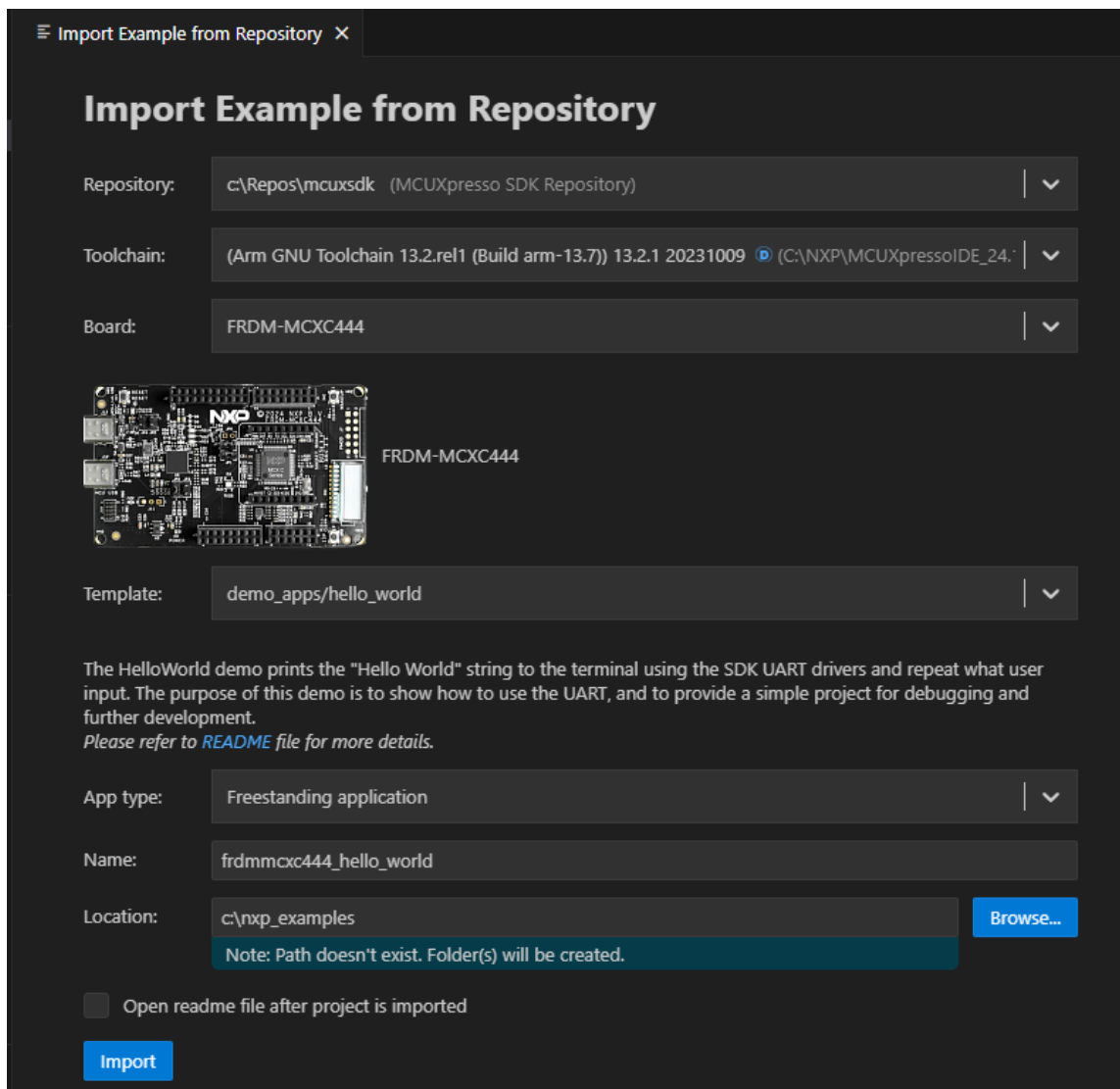
1. Click **Import Example from Repository** from the **QUICKSTART PANEL**



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:

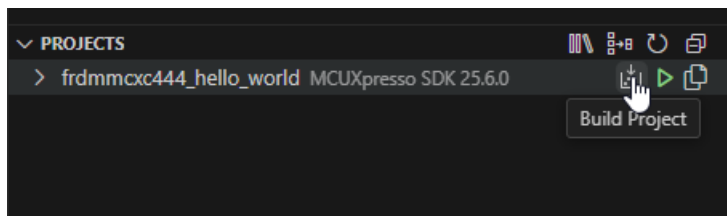
- Imported to user-defined location
- Independent of SDK location

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

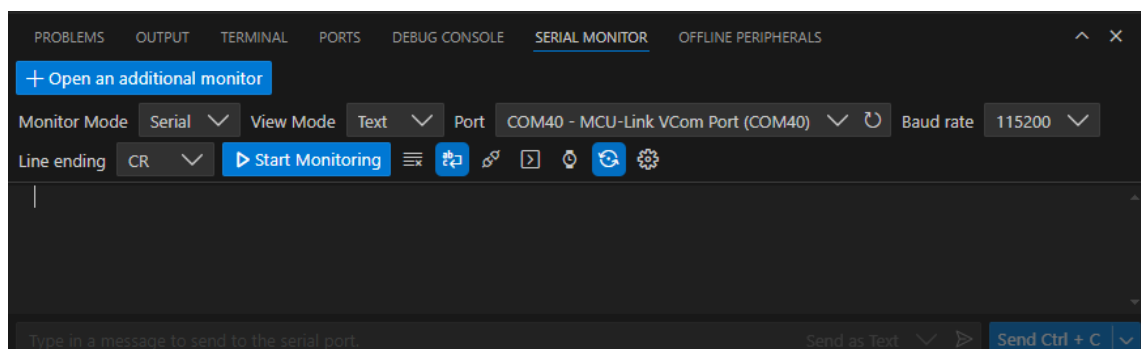


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

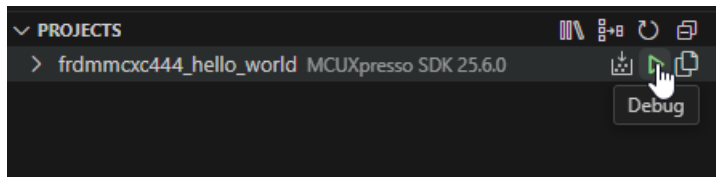
1. Open **Serial Monitor** from VS Code's integrated terminal



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

Debug Session

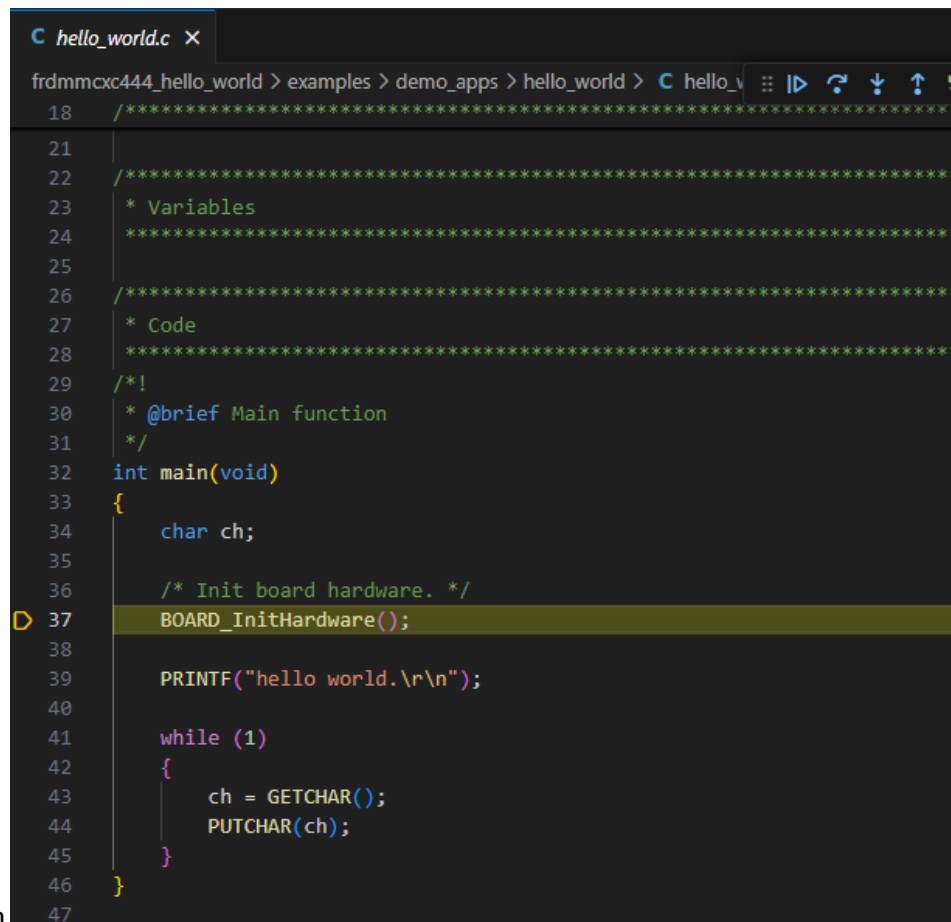
1. Navigate to **PROJECTS** view
2. Click the play button to initiate a debug session



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

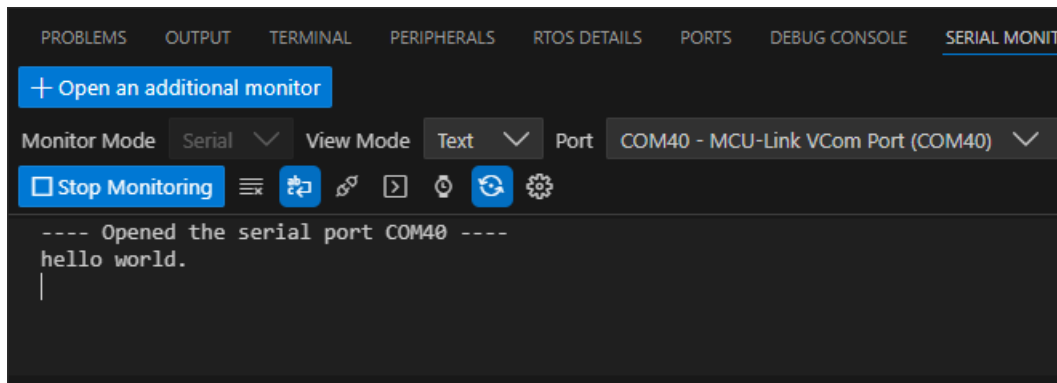
Debug Controls Use the debug controls to manage execution:

- **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.

De- vel- op- ment board	MCU devices				
FRDM-MCXA	MCXA577VLL,	MCXA577VLQ,	MCXA577VPN,	MCXA286VLL,	MCXA286VLQ,
	MCXA286VPN,	MCXA287VLL,	MCXA287VLQ,	MCXA287VPN,	MCXA456VLL,
	MCXA456VLQ,	MCXA456VPN,	MCXA457VLL,	MCXA457VLQ,	MCXA457VPN,
	MCXA536VLL,	MCXA536VLQ,	MCXA536VPN,	MCXA537VLL,	MCXA537VLQ,
	MCXA537VPN,	MCXA556VPN,	MCXA566VLL,	MCXA566VLQ,	MCXA566VPN,
	MCXA567VLL, MCXA567VLQ, MCXA567VPN				

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.

coreHTTP coreHTTP

mbedTLS mbedtls SSL/TLS library v3.x

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

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

MCU Boot Open source MCU Bootloader.

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

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

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

LVGL LVGL Open Source Graphics Library

llhttp HTTP parser llhttp

LittleFS LittleFS filesystem stack

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

Metering Metering Library

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

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

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

AOI

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

CDOG

[2.1.3]

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

[2.1.2]

- Support multiple IRQs
- Fix default CONTROL values

[2.1.1]

- Remove bit CONTROL[CONTROL_CTRL].

[2.1.0]

- Rename CWT to CDOG.

[2.0.2]

- Fix MISRA-2012 issues.

[2.0.1]

- Fix doxygen issues.

[2.0.0]

- Initial version.
-

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.
-

DAC**[2.1.3]**

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

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.
-

MCX_ENET

[2.2.1]

- Bug Fixes
 - Fixed ENET_EnableInterrupts to preserve already enabled interrupts.
 - Fixed ENET_DisableInterrupts to correctly manage interrupt summary bits.
 - Fixed IRQ handler to check interrupt enable status before processing events.

[2.2.0]

- New features
 - Support for the version of peripheral with reduced features like a single DMA channel.

[2.1.4]

- Bug Fixes
 - Fixed ENET_GetMacAddr address byte order not matching ENET_SetMacAddr.

[2.1.3]

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

[2.1.2]

- New features
 - Added hardware checksum acceleration support.

[2.1.1]

- Bug Fixes
 - Fixed the issue that free wrong buffer address when one frame stores in multiple buffers and memory pool is not enough to allocate these buffers to receive one complete frame.
 - Fixed the issue that ENET_DropFrame checks the buffer descriptor flag after it has been re-initialized.
 - Fixed the ENET_GetRxFrame FCS calculation issue.
 - Fixed the issue that there's no valid error type in the return structure when Rx error bit is set.

[2.1.0]

- New Features
 - Added the VLAN control setting APIs in the driver.

[2.0.1]

- Bug Fixes
 - Fixed the issue that enable/disable interrupt APIs miss part of configuration.

[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.
-

ESPI

[2.0.0]

- Initial version.
-

EWM

[2.0.4]

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the hard fault in EWM_Deinit.

[2.0.0]

- Initial version.
-

FLEXIO

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
 - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

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

[2.1.0]

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

[2.0.4]

- Bug Fixes
 - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.2]

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

[2.0.1]

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

FLEXIO_I2C

[2.6.2]

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

[2.6.1]

- Bug Fixes
 - Fixed coverity issues

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S_EDMA

[2.1.9]

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

[2.1.8]

- Improvements
 - Applied EDMA ERRATA 51327.
-

FLEXIO_MCU_LCD

[2.3.0]

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

[2.2.0]

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

[2.1.0]

- New Features
 - Supported transmit only data without command.

[2.0.8]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

FLEXIO_MCU_LCD_EDMA

[2.0.6]

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

[2.0.5]

- New Features
 - Supported transmit only data without command.

[2.0.4]

- Bug Fixes
 - Fixed MISRA 10.4 issues.

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

FLEXIO_SPI

[2.4.3]

- Improvements
 - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

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

[2.4.1]

- Bug Fixes
 - Fixed coverity issues

[2.4.0]

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

[2.3.5]

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

[2.3.4]

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

[2.3.3]

- Bugfixes
 - Fixed cs-continuous mode.

[2.3.2]

- Improvements
 - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

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

[2.3.0]

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

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

[2.2.1]

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

[2.2.0]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

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

[2.1.0]

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

FLEXIO_UART

[2.6.4]

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

[2.6.3]

- Bug Fixes
 - Fixed coverity issues

[2.6.2]

- Bug Fixes
 - Fixed coverity issues

[2.6.1]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

- Improvements
 - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_UART_EDMA

[2.3.1]

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

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0
-

FLEXSPI

[2.9.0]

- New Features
 - Added FlexSPI root clock divider configuration.
- Bug Fixes
 - Add explicit boolean to integer conversions (? 1U : 0U) for all enable flags
 - Add assertions to prevent unsigned integer overflow in buffer size calculations
 - Add assertions to prevent unsigned integer underflow in watermark calculations
 - Add safe type conversion for uint32_t to uint8_t operations
 - Add bounds checking assertion for LUT index operations
 - Follow INT30-C and INT31-C CERT coding standards

[2.8.1]

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

[2.8.0]

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

[2.7.0]

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

[2.6.4]

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

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.5]

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

[2.3.4]

- Bug Fixes
 - Updated `flexspi_config_t` structure and `FlexSPI_Init` to support new feature `FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_COMBINATION`.

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

FLEXSPI EDMA Driver

[2.3.3]

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

[2.3.2]

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

[2.0.2]

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

[2.0.0]

- Initial version.
-

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.
-

GLIKEY**[2.0.1]**

- Implement INIT state recovery from the LOCKED state after a reset when the previous index was locked.

[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_GetPinsDMARequestFlags` with `GPIO_ISFR_COUNT <= 1`.

[2.7.3]

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

[2.7.2]

- New Features
 - Support devices without PORT module.

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
 - Removed deprecated APIs.

[2.3.0]

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

[2.2.1]

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

[2.1.1]

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

[2.1.0]

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

I3C**[2.14.4]**

- Improvements
 - Fixed MSG violations.

[2.14.3]

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

[2.14.2]

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

[2.14.1]

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

[2.14.0]

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

[2.13.1]

- Bug Fixes
 - Disabled Rx auto-termination in repeated start interrupt event while transfer API doesn’t enable it.
 - Waited the completion event after loading all Tx data in Tx FIFO.
- Improvements

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

[2.13.0]

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

[2.12.0]

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

[2.11.1]

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

[2.11.0]

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

[2.10.8]

- Improvements
 - Support more instances.

[2.10.7]

- Improvements
 - Fixed the potential compile warning.

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- New features
 - Added I3C extended IBI data support.

[2.9.0]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.4]

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

[2.5.3]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH and FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

I3C_EDMA

[2.2.12]

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

[2.2.11]

- Improvements
 - Fixed Coverity CERT-C violations.

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Bug Fixes
 - Fixed MISRA issue rule 11.6.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
 - Deleted legacy IBI data request code.

[2.1.0]

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

[2.0.1]

- Bug Fixes
 - Fixed MISRA issue rule 9.1.

[2.0.0]

- Initial version.
-

INPUTMUX**[2.0.10]**

- Bug Fixes
 - Fixed CERT-C violations.

[2.0.9]

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

[2.0.8]

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

[2.0.7]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Support channel mux setting in INPUTMUX_EnableSignal().

[2.0.0]

- Initial version.
-

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.
-

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 LPSPi_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 LPSPIn_CR[RST] bit.

[2.5.0]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.1]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Bug Fixes
 - Fixed the bug that the clock source should be divided by the PRESCALE setting in LP-SPI_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 controled by wrong feature macro.

[2.8.1]

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

[2.8.0]

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

[2.7.7]

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

[2.7.6]

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

[2.7.5]

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

[2.7.4]

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

[2.7.3]

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

[2.7.2]

- Bug Fix
 - Fixed the bug that the OSR calculation error when 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
-

OSTIMER

[2.2.6]

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

[2.2.5]

- Improvements
 - Support binary encoded ostimer.

[2.2.4]

- Bug Fixes
 - Fixed CERT INT31-C violations.

[2.2.3]

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

[2.2.2]

- Improvements
 - Support devices with different OSTIMER instance name.

[2.2.1]

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

[2.2.0]

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

[2.1.2]

- Bug Fixes
 - Fixed MISRA-2012 rule 10.8.

[2.1.1]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

- Improvements
 - Added support for OSTIMER0

[2.0.1]

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

[2.0.0]

- Initial version.
-

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.
-

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.
-

TENBASET_PHY

[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.
-

UTICK

[2.0.5]

- Improvements
 - Improved for SOC RW610.

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.
-

VREF

[2.5.0]

- New features
 - Add VREF_SetAdcNegChannelRef() API for ADC negative channel control
 - Add VREF_SetAdcPosChannelRef() API for ADC positive channel control
 - Add conditional LPBGEN_MASK setting based on feature availability
 - Add IBIAS_EN_MASK setting for devices with IBIAS_EN feature

[2.4.1]

- Improvements
 - Improve doxygen documentation.

[2.4.0]

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

[2.3.0]

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

[2.2.2]

- Bug Fixes
 - Fixed typo in vref_buffer_mode_t enumerator.

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.0]

- Initial version.
-

WAKETIMER**[2.0.1]**

- Bug Fix
 - MISRA C-2012 issue fixed: rule 8.2, 8.8, 10.1, 10.4.

[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.

[MCXA577](#)

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

MCXA577

2.1 AOI: Crossbar AND/OR/INVERT Driver

`void AOI_Init(AOI_Type *base)`

Initializes an AOI instance for operation.

This function un-gates the AOI clock.

Parameters

- `base` – AOI peripheral address.

`void AOI_Deinit(AOI_Type *base)`

Deinitializes an AOI instance for operation.

This function shutdowns AOI module.

Parameters

- `base` – AOI peripheral address.

`void AOI_GetEventLogicConfig(AOI_Type *base, aoi_event_t event, aoi_event_config_t *config)`

Gets the Boolean evaluation associated.

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;

AOI_GetEventLogicConfig(AOI, kAOI_Event0, &demoEventLogicStruct);
```

Parameters

- `base` – AOI peripheral address.
- `event` – Index of the event which will be set of type `aoi_event_t`.
- `config` – Selected input configuration .

`void AOI_SetEventLogicConfig(AOI_Type *base, aoi_event_t event, const aoi_event_config_t *eventConfig)`

Configures an AOI event.

This function configures an AOI event according to the `aoiEventConfig` structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event.

Example:

```
aoi_event_config_t demoEventLogicStruct;

demoEventLogicStruct.PT0AC = kAOI_InvInputSignal;
demoEventLogicStruct.PT0BC = kAOI_InputSignal;
demoEventLogicStruct.PT0CC = kAOI_LogicOne;
demoEventLogicStruct.PT0DC = kAOI_LogicOne;

demoEventLogicStruct.PT1AC = kAOI_LogicZero;
demoEventLogicStruct.PT1BC = kAOI_LogicOne;
demoEventLogicStruct.PT1CC = kAOI_LogicOne;
demoEventLogicStruct.PT1DC = kAOI_LogicOne;

demoEventLogicStruct.PT2AC = kAOI_LogicZero;
demoEventLogicStruct.PT2BC = kAOI_LogicOne;
demoEventLogicStruct.PT2CC = kAOI_LogicOne;
demoEventLogicStruct.PT2DC = kAOI_LogicOne;

demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
demoEventLogicStruct.PT3DC = kAOI_LogicOne;

AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

Parameters

- base – AOI peripheral address.
- event – Event which will be configured of type `aoi_event_t`.
- eventConfig – Pointer to type `aoi_event_config_t` structure. The user is responsible for filling out the members of this structure and passing the pointer to this function.

FSL_AOI_DRIVER_VERSION

Version 2.0.2.

enum _aoi_input_config

AOI input configurations.

The selection item represents the Boolean evaluations.

Values:

enumerator kAOI_LogicZero

Forces the input to logical zero.

enumerator kAOI_InputSignal

Passes the input signal.

enumerator kAOI_InvInputSignal

Inverts the input signal.

enumerator kAOI_LogicOne

Forces the input to logical one.

enum _aoi_event

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

Values:

enumerator kAOI_Event0

Event 0 index

enumerator kAOI_Event1

Event 1 index

enumerator kAOI_Event2

Event 2 index

enumerator kAOI_Event3

Event 3 index

typedef enum *_aoi_input_config* aoi_input_config_t

AOI input configurations.

The selection item represents the Boolean evaluations.

typedef enum *_aoi_event* aoi_event_t

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

typedef struct *_aoi_event_config* aoi_event_config_t

AOI event configuration structure.

Defines structure *_aoi_event_config* and use the *AOI_SetEventLogicConfig()* function to make whole event configuration.

AOI

AOI peripheral address

struct *_aoi_event_config*

#include <fsl_aoi.h> AOI event configuration structure.

Defines structure *_aoi_event_config* and use the *AOI_SetEventLogicConfig()* function to make whole event configuration.

Public Members

aoi_input_config_t PT0AC

Product term 0 input A

aoi_input_config_t PT0BC

Product term 0 input B

aoi_input_config_t PT0CC

Product term 0 input C

aoi_input_config_t PT0DC

Product term 0 input D

aoi_input_config_t PT1AC

Product term 1 input A

aoi_input_config_t PT1BC

Product term 1 input B

aoi_input_config_t PT1CC

Product term 1 input C

aoi_input_config_t PT1DC

Product term 1 input D

aoi_input_config_t PT2AC

Product term 2 input A

aoi_input_config_t PT2BC
Product term 2 input B

aoi_input_config_t PT2CC
Product term 2 input C

aoi_input_config_t PT2DC
Product term 2 input D

aoi_input_config_t PT3AC
Product term 3 input A

aoi_input_config_t PT3BC
Product term 3 input B

aoi_input_config_t PT3CC
Product term 3 input C

aoi_input_config_t PT3DC
Product term 3 input D

2.2 CDOG

status_t CDOG_Init(CDOG_Type *base, *cdog_config_t* *conf)
Initialize CDOG.

This function initializes CDOG block and setting.

Parameters

- base – CDOG peripheral base address
- conf – CDOG configuration structure

Returns

Status of the init operation

void CDOG_Deinit(CDOG_Type *base)
Deinitialize CDOG.

This function deinitializes CDOG secure counter.

Parameters

- base – CDOG peripheral base address

void CDOG_GetDefaultConfig(*cdog_config_t* *conf)
Sets the default configuration of CDOG.

This function initialize CDOG config structure to default values.

Parameters

- conf – CDOG configuration structure

void CDOG_Stop(CDOG_Type *base, uint32_t stop)
Stops secure counter and instruction timer.

This function stops instruction timer and secure counter. This also change state of CDOG to IDLE.

Parameters

- base – CDOG peripheral base address

- stop – expected value which will be compared with value of secure counter

void CDOG_Start(CDOG_Type *base, uint32_t reload, uint32_t start)

Sets secure counter and instruction timer values.

This function sets value in RELOAD and START registers for instruction timer and secure counter

Parameters

- base – CDOG peripheral base address
- reload – reload value
- start – start value

void CDOG_Check(CDOG_Type *base, uint32_t check)

Checks secure counter.

This function compares stop value in handler with secure counter value by writting to RELOAD refister.

Parameters

- base – CDOG peripheral base address
- check – expected (stop) value

void CDOG_Set(CDOG_Type *base, uint32_t stop, uint32_t reload, uint32_t start)

Sets secure counter and instruction timer values.

This function sets value in STOP, RELOAD and START registers for instruction timer and secure counter.

Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter
- reload – reload value for instruction timer
- start – start value for secure timer

void CDOG_Add(CDOG_Type *base, uint32_t add)

Add value to secure counter.

This function add specified value to secure counter.

Parameters

- base – CDOG peripheral base address.
- add – Value to be added.

void CDOG_Add1(CDOG_Type *base)

Add 1 to secure counter.

This function add 1 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add16(CDOG_Type *base)

Add 16 to secure counter.

This function add 16 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add256(CDOG_Type *base)

Add 256 to secure counter.

This function add 256 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub(CDOG_Type *base, uint32_t sub)

brief Subtract value to secure counter

This function subtract specified value to secure counter.

param base CDOG peripheral base address. param sub Value to be subtracted.

void CDOG_Sub1(CDOG_Type *base)

Subtract 1 from secure counter.

This function subtract specified 1 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub16(CDOG_Type *base)

Subtract 16 from secure counter.

This function subtract specified 16 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub256(CDOG_Type *base)

Subtract 256 from secure counter.

This function subtract specified 256 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_WritePersistent(CDOG_Type *base, uint32_t value)

Set the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.
- value – The value to be written.

uint32_t CDOG_ReadPersistent(CDOG_Type *base)

Get the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.

Returns

The persistent word.

FSL_CDOG_DRIVER_VERSION

Defines CDOG driver version 2.1.3.

Change log:

- Version 2.1.3
 - Re-design multiple instance IRQs and Clocks
 - Add fix for RESTART command errata

- Version 2.1.2
 - Support multiple IRQs
 - Fix default CONTROL values
- Version 2.1.1
 - Remove bit CONTROL[CONTROL_CTRL]
- Version 2.1.0
 - Rename CWT to CDOG
- Version 2.0.2
 - Fix MISRA-2012 issues
- Version 2.0.1
 - Fix doxygen issues
- Version 2.0.0
 - initial version

enum __cdog_debug_Action_ctrl_enum

Values:

enumerator kCDOG_DebugHaltCtrl_Run

enumerator kCDOG_DebugHaltCtrl_Pause

enum __cdog_irq_pause_ctrl_enum

Values:

enumerator kCDOG_IrqPauseCtrl_Run

enumerator kCDOG_IrqPauseCtrl_Pause

enum __cdog_fault_ctrl_enum

Values:

enumerator kCDOG_FaultCtrl_EnableReset

enumerator kCDOG_FaultCtrl_EnableInterrupt

enumerator kCDOG_FaultCtrl_NoAction

enum __code_lock_ctrl_enum

Values:

enumerator kCDOG_LockCtrl_Lock

enumerator kCDOG_LockCtrl_Unlock

typedef uint32_t secure_counter_t

SC_ADD(add)

SC_ADD1

SC_ADD16

SC_ADD256

SC_SUB(sub)

SC_SUB1

SC_SUB16

SC_SUB256

SC_CHECK(val)

```
struct cdog_config_t
    #include <fsl_cdog.h>
```

2.3 Clock Driver

enum _clock_ip_name

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

enumerator kCLOCK_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_GateCTIMER2

Clock gate name: CTIMER2

enumerator kCLOCK_GateCTIMER3

Clock gate name: CTIMER3

enumerator kCLOCK_GateCTIMER4

Clock gate name: CTIMER4

enumerator kCLOCK_GateUTICK0

Clock gate name: UTICK0

enumerator kCLOCK_GateWWDT0

Clock gate name: WWDT0

enumerator kCLOCK_GateWWDT1

Clock gate name: WWDT1

enumerator kCLOCK_GateDMA0

Clock gate name: DMA0

enumerator kCLOCK_GateDMA1

Clock gate name: DMA1

enumerator kCLOCK_GateAOI0

Clock gate name: AOI0

enumerator kCLOCK_GateCRC0

Clock gate name: CRC0

enumerator kCLOCK_GateEIM0

Clock gate name: EIM0

enumerator kCLOCK_GateERM0
Clock gate name: ERM0

enumerator kCLOCK_GateFLEXIO0
Clock gate name: FLEXIO0

enumerator kCLOCK_GateLPI2C0
Clock gate name: LPI2C0

enumerator kCLOCK_GateLPI2C1
Clock gate name: LPI2C1

enumerator kCLOCK_GateLPI2C2
Clock gate name: LPI2C2

enumerator kCLOCK_GateLPI2C3
Clock gate name: LPI2C3

enumerator kCLOCK_GateLPI2C4
Clock gate name: LPI2C4

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_GateLPUART4
Clock gate name: LPUART4

enumerator kCLOCK_GateLPUART5
Clock gate name: LPUART5

enumerator kCLOCK_GateOSTIMER0
Clock gate name: OSTIMER0

enumerator kCLOCK_GateLPSPI0
Clock gate name: LPSPI0

enumerator kCLOCK_GateLPSPI1
Clock gate name: LPSPI1

enumerator kCLOCK_GateLPSPI2
Clock gate name: LPSPI2

enumerator kCLOCK_GateLPSPI3
Clock gate name: LPSPI3

enumerator kCLOCK_GateLPSPI4
Clock gate name: LPSPI4

enumerator kCLOCK_GateLPSPI5
Clock gate name: LPSPI5

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_GatePORT4
Clock gate name: PORT4

enumerator kCLOCK_GatePORT5
Clock gate name: PORT5

enumerator kCLOCK_GateADC0
Clock gate name: ADC0

enumerator kCLOCK_GateADC1
Clock gate name: ADC1

enumerator kCLOCK_GateCMP0
Clock gate name: CMP0

enumerator kCLOCK_GateDAC0
Clock gate name: DAC0

enumerator kCLOCK_GateDAC1
Clock gate name: DAC1

enumerator kCLOCK_GateVREF0
Clock gate name: VREF0

enumerator kCLOCK_GateTSI0
Clock gate name: TSI0

enumerator kCLOCK_GateI3C0
Clock gate name: I3C0

enumerator kCLOCK_GateI3C1
Clock gate name: I3C1

enumerator kCLOCK_GateI3C2
Clock gate name: I3C2

enumerator kCLOCK_GateI3C3
Clock gate name: I3C3

enumerator kCLOCK_GateFLEXCAN0
Clock gate name: FLEXCAN0

enumerator kCLOCK_GateFLEXCAN1
Clock gate name: FLEXCAN1

enumerator kCLOCK_GateE1588
Clock gate name: E1588

enumerator kCLOCK_GateRMII
Clock gate name: RMII

enumerator kCLOCK_GateENET0
Clock gate name: ENET0

enumerator kCLOCK_GateTENBASET_PHY0

 Clock gate name: TENBASET_PHY0

enumerator kCLOCK_GateFLEXSPI0

 Clock gate name: FLEXSPI0

enumerator kCLOCK_GateSPIFILTER0

 Clock gate name: SPIFILTER0

enumerator kCLOCK_GateESPI0

 Clock gate name: ESPI0

enumerator kCLOCK_GateUSBHS

 Clock gate name: USBHS

enumerator kCLOCK_GateUSBHS_PHY

 Clock gate name: USBHS_PHY

enumerator kCLOCK_GateEWM0

 Clock gate name: EWM0

enumerator kCLOCK_GateRAMA

 Clock gate name: RAMA

enumerator kCLOCK_GateRAMB

 Clock gate name: RAMB

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_GateGPIO4

 Clock gate name: GPIO4

enumerator kCLOCK_GateGPIO5

 Clock gate name: GPIO5

enumerator kCLOCK_GateROMC

 Clock gate name: ROMC

enumerator kCLOCK_GateSMARTDMA0

 Clock gate name: SMARTDMA0

enumerator kCLOCK_GateSECCON

 Clock gate name: SECCON

enumerator kCLOCK_GateGLIKEY0

 Clock gate name: GLIKEY0

enumerator kCLOCK_GateTDET0

 Clock gate name: TDET0

enumerator kCLOCK_GatePKC0

 Clock gate name: PKC0

enumerator kCLOCK_GateSGI0
Clock gate name: SGI0

enumerator kCLOCK_GateTRNG0
Clock gate name: TRNG0

enumerator kCLOCK_GateUDF0
Clock gate name: UDF0

enumerator kCLOCK_GateDGDET0
Clock gate name: DGDET0

enumerator kCLOCK_GateITRC0
Clock gate name: ITRC0

enumerator kCLOCK_GateATX0
Clock gate name: ATX0

enumerator kCLOCK_GateMTR
Clock gate name: MTR

enumerator kCLOCK_GateTCU
Clock gate name: TCU

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 divided by 2

enumerator kCLOCK_SLOW_CLK
SYSTEM clock divided by 6

enumerator kCLOCK_ExtClk
External Clock

enumerator kCLOCK_Fro16K
FRO16K

enumerator kCLOCK_Fro12M
FRO12M

enumerator kCLOCK_Fro12MDiv
FRO12MDiv

enumerator kCLOCK_FroHf
FROHF

enumerator kCLOCK_FroHfDiv

Divided by FROHF

enumerator kCLOCK_Clk48M

CLK48M

enumerator kCLOCK_Pll1Clk

Pll1Clk

enumerator kCLOCK_Pll1ClkDiv

Pll1ClkDiv

enumerator kCLOCK_Clk1M

CLK1M

enumerator kCLOCK_UsbPll

UsbPll

enumerator kCLOCK_UsbPfdClk

UsbPfdClk

enumerator kCLOCK_Clk16K0

CLK16K[0]

enumerator kCLOCK_Clk16K1

CLK16K[1]

enumerator kCLOCK_Clk16K2

CLK16K[2]

enumerator kCLOCK_Osc32K0

OSC32K[0]

enumerator kCLOCK_Osc32K1

OSC32K[1]

enumerator kCLOCK_Osc32K2

OSC32K[2]

enumerator kCLOCK_LpOsc

Low Power Oscillator

enum _clock_select_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_SelI3C0_FCLK

I3C0_FCLK clock selection

enumerator kCLOCK_SelI3C1_FCLK

I3C1_FCLK clock selection

enumerator kCLOCK_SelCTIMER0

CTIMER0 clock selection

enumerator kCLOCK_SelCTIMER1

CTIMER1 clock selection

enumerator kCLOCK_SelCTIMER2

CTIMER2 clock selection

enumerator kCLOCK_SelCTIMER3
CTIMER3 clock selection

enumerator kCLOCK_SelCTIMER4
CTIMER4 clock selection

enumerator kCLOCK_SelWWDT1
WWDT1 clock selection

enumerator kCLOCK_SelE1588
E1588 clock selection

enumerator kCLOCK_SelRMII
RMII clock selection

enumerator kCLOCK_SelESPI0
ESPI0 clock selection

enumerator kCLOCK_SelFLEXSPI0
FLEXSPI0 clock selection

enumerator kCLOCK_SelLPSPi2
LPSPi2 clock selection

enumerator kCLOCK_SelLPSPi3
LPSPi3 clock selection

enumerator kCLOCK_SelLPSPi4
LPSPi4 clock selection

enumerator kCLOCK_SelLPSPi5
LPSPi5 clock selection

enumerator kCLOCK_SelTENBASET_PHY0
TENBASET_PHY0 clock selection

enumerator kCLOCK_SelUSBHS
USBHS clock selection

enumerator kCLOCK_SelUSBHS_PHY
USBHS_PHY clock selection

enumerator kCLOCK_SelFLEXIO0
FLEXIO0 clock selection

enumerator kCLOCK_SelLPI2C0
LPI2C0 clock selection

enumerator kCLOCK_SelLPI2C1
LPI2C1 clock selection

enumerator kCLOCK_SelLPSPi0
LPSPi0 clock selection

enumerator kCLOCK_SelLPSPi1
LPSPi1 clock selection

enumerator kCLOCK_SelI3C2_FCLK
I3C2_FCLK 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_SelLPUART4
LPUART4 clock selection

enumerator kCLOCK_SelLPTMR0
LPTMR0 clock selection

enumerator kCLOCK_SelOSTIMER0
OSTIMER0 clock selection

enumerator kCLOCK_SelADC
ADCx clock selection

enumerator kCLOCK_SelCMP0_RR
CMP0_RR clock selection

enumerator kCLOCK_SelDAC0
DAC0 clock selection

enumerator kCLOCK_SelDAC1
DAC1 clock selection

enumerator kCLOCK_SelTSI0
TSI0 clock selection

enumerator kCLOCK_SelFLEXCAN0
FLEXCAN0 clock selection

enumerator kCLOCK_SelFLEXCAN1
FLEXCAN1 clock selection

enumerator kCLOCK_SelLPI2C2
LPI2C2 clock selection

enumerator kCLOCK_SelLPI2C3
LPI2C3 clock selection

enumerator kCLOCK_SelLPI2C4
LPI2C4 clock selection

enumerator kCLOCK_SelLPUART5
LPUART5 clock selection

enumerator kCLOCK_SelI3C3_FCLK
I3C3_FCLK 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 kOSC_32K_to_MAIN_CLK

Attach OSC_32K[1] to MAIN_CLK.

enumerator kPll1Clk_to_MAIN_CLK

Attach Pll1Clk to MAIN_CLK.

enumerator kUSB_PFD_to_MAIN_CLK

Attach Pll1Clk to MAIN_CLK.

enumerator kNONE_to_MAIN_CLK

Attach NONE to MAIN_CLK.

enumerator kFRO_LF_DIV_to_I3C0FCLK

Attach FRO_LF_DIV to I3C0FCLK.

enumerator kFRO_HF_DIV_to_I3C0FCLK

Attach FRO_HF_DIV to I3C0FCLK.

enumerator kCLK_IN_to_I3C0FCLK

Attach CLK_IN to I3C0FCLK.

enumerator kCLK_1M_to_I3C0FCLK

Attach CLK_1M to I3C0FCLK.

enumerator kPll1ClkDiv_to_I3C0FCLK

Attach Pll1ClkDiv to I3C0FCLK.

enumerator kNONE_to_I3C0FCLK

Attach NONE to I3C0FCLK.

enumerator kFRO_LF_DIV_to_I3C1FCLK

Attach FRO_LF_DIV to I3C1FCLK.

enumerator kFRO_HF_DIV_to_I3C1FCLK

Attach FRO_HF_DIV to I3C1FCLK.

enumerator kCLK_IN_to_I3C1FCLK

Attach CLK_IN to I3C1FCLK.

enumerator kCLK_1M_to_I3C1FCLK

Attach CLK_1M to I3C1FCLK.

enumerator kPll1ClkDiv_to_I3C1FCLK
Attach Pll1ClkDiv to I3C1FCLK.

enumerator kNONE_to_I3C1FCLK
Attach NONE to I3C1FCLK.

enumerator kFRO_LF_DIV_to_I3C2FCLK
Attach FRO_LF_DIV to I3C2FCLK.

enumerator kFRO_HF_DIV_to_I3C2FCLK
Attach FRO_HF_DIV to I3C2FCLK.

enumerator kCLK_IN_to_I3C2FCLK
Attach CLK_IN to I3C2FCLK.

enumerator kCLK_1M_to_I3C2FCLK
Attach CLK_1M to I3C2FCLK.

enumerator kPll1ClkDiv_to_I3C2FCLK
Attach Pll1ClkDiv to I3C2FCLK.

enumerator kNONE_to_I3C2FCLK
Attach NONE to I3C2FCLK.

enumerator kFRO_LF_DIV_to_I3C3FCLK
Attach FRO_LF_DIV to I3C3FCLK.

enumerator kFRO_HF_DIV_to_I3C3FCLK
Attach FRO_HF_DIV to I3C3FCLK.

enumerator kCLK_IN_to_I3C3FCLK
Attach CLK_IN to I3C3FCLK.

enumerator kCLK_1M_to_I3C3FCLK
Attach CLK_1M to I3C3FCLK.

enumerator kPll1ClkDiv_to_I3C3FCLK
Attach Pll1ClkDiv to I3C3FCLK.

enumerator kNONE_to_I3C3FCLK
Attach NONE to I3C3FCLK.

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 kLP_OSC_to_CTIMER0
Attach LP_OSC to CTIMER0.

enumerator kCLK_1M_to_CTIMER0
Attach CLK_1M to CTIMER0.

enumerator kPll1ClkDiv_to_CTIMER0
Attach Pll1ClkDiv 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 kLP_OSC_to_CTIMER1
Attach LP_OSC to CTIMER1.

enumerator kCLK_1M_to_CTIMER1
Attach CLK_1M to CTIMER1.

enumerator kPll1ClkDiv_to_CTIMER1
Attach Pll1ClkDiv to CTIMER1.

enumerator kNONE_to_CTIMER1
Attach NONE to CTIMER1.

enumerator kFRO_LF_DIV_to_CTIMER2
Attach FRO_LF_DIV to CTIMER2.

enumerator kFRO_HF_to_CTIMER2
Attach FRO_HF to CTIMER2.

enumerator kCLK_IN_to_CTIMER2
Attach CLK_IN to CTIMER2.

enumerator kLP_OSC_to_CTIMER2
Attach LP_OSC to CTIMER2.

enumerator kCLK_1M_to_CTIMER2
Attach CLK_1M to CTIMER2.

enumerator kPll1ClkDiv_to_CTIMER2
Attach Pll1ClkDiv to CTIMER2.

enumerator kNONE_to_CTIMER2
Attach NONE to CTIMER2.

enumerator kFRO_LF_DIV_to_CTIMER3
Attach FRO_LF_DIV to CTIMER3.

enumerator kFRO_HF_to_CTIMER3
Attach FRO_HF to CTIMER3.

enumerator kCLK_IN_to_CTIMER3
Attach CLK_IN to CTIMER3.

enumerator kLP_OSC_to_CTIMER3
Attach LP_OSC to CTIMER3.

enumerator kCLK_1M_to_CTIMER3
Attach CLK_1M to CTIMER3.

enumerator kPll1ClkDiv_to_CTIMER3
Attach Pll1ClkDiv to CTIMER3.

enumerator kNONE_to_CTIMER3
Attach NONE to CTIMER3.

enumerator kFRO_LF_DIV_to_CTIMER4

Attach FRO_LF_DIV to CTIMER4.

enumerator kFRO_HF_to_CTIMER4

Attach FRO_HF to CTIMER4.

enumerator kCLK_IN_to_CTIMER4

Attach CLK_IN to CTIMER4.

enumerator kLP_OSC_to_CTIMER4

Attach LP_OSC to CTIMER4.

enumerator kCLK_1M_to_CTIMER4

Attach CLK_1M to CTIMER4.

enumerator kPll1ClkDiv_to_CTIMER4

Attach Pll1ClkDiv to CTIMER4.

enumerator kNONE_to_CTIMER4

Attach NONE to CTIMER4.

enumerator kFRO_LF_DIV_to_FLEXIO0

Attach FRO_LF_DIV to FLEXIO0.

enumerator kFRO_HF_to_FLEXIO0

Attach FRO_HF to FLEXIO0.

enumerator kCLK_IN_to_FLEXIO0

Attach CLK_IN to FLEXIO0.

enumerator kCLK_1M_to_FLEXIO0

Attach CLK_1M to FLEXIO0.

enumerator kPll1ClkDiv_to_FLEXIO0

Attach Pll1ClkDiv to FLEXIO0.

enumerator kNONE_to_FLEXIO0

Attach NONE to FLEXIO0.

enumerator kFRO_HF_to_FLEXCAN0

Attach FRO_HF to FLEXCAN0.

enumerator kFRO_HF_DIV_to_FLEXCAN0

Attach FRO_HF_DIV to FLEXCAN0.

enumerator kCLK_IN_to_FLEXCAN0

Attach CLK_IN to FLEXCAN0.

enumerator kUSB_PLL_to_FLEXCAN0

Attach USB_PLL to FLEXCAN0.

enumerator kPll1ClkDiv_to_FLEXCAN0

Attach Pll1ClkDiv to FLEXCAN0.

enumerator kNONE_to_FLEXCAN0

Attach NONE to FLEXCAN0.

enumerator kFRO_HF_to_FLEXCAN1

Attach FRO_HF to FLEXCAN1.

enumerator kFRO_HF_DIV_to_FLEXCAN1

Attach FRO_HF_DIV to FLEXCAN1.

enumerator kCLK_IN_to_FLEXCAN1
Attach CLK_IN to FLEXCAN1.

enumerator kUSB_PLL_to_FLEXCAN1
Attach USB_PLL to FLEXCAN1.

enumerator kPll1ClkDiv_to_FLEXCAN1
Attach Pll1ClkDiv to FLEXCAN1.

enumerator kNONE_to_FLEXCAN1
Attach NONE to FLEXCAN1.

enumerator kFRO_LF_DIV_to_DAC0
Attach FRO_LF_DIV to DAC0.

enumerator kFRO_HF_DIV_to_DAC0
Attach FRO_HF_DIV to DAC0.

enumerator kCLK_IN_to_DAC0
Attach CLK_IN to DAC0.

enumerator kCLK_1M_to_DAC0
Attach CLK_1M to DAC0.

enumerator kPll1ClkDiv_to_DAC0
Attach Pll1ClkDiv to DAC0.

enumerator kNONE_to_DAC0
Attach NONE to DAC0.

enumerator kFRO_LF_DIV_to_DAC1
Attach FRO_LF_DIV to DAC1.

enumerator kFRO_HF_DIV_to_DAC1
Attach FRO_HF_DIV to DAC1.

enumerator kCLK_IN_to_DAC1
Attach CLK_IN to DAC1.

enumerator kCLK_1M_to_DAC1
Attach CLK_1M to DAC1.

enumerator kPll1ClkDiv_to_DAC1
Attach Pll1ClkDiv to DAC1.

enumerator kNONE_to_DAC1
Attach NONE to DAC1.

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 kPll1ClkDiv_to_LPI2C0
Attach Pll1ClkDiv to LPI2C0.

enumerator kNONE_to_LPI2C0
Attach NONE to LPI2C0.

enumerator kFRO_LF_DIV_to_LPI2C1
Attach FRO_LF_DIV to LPI2C1.

enumerator kFRO_HF_DIV_to_LPI2C1
Attach FRO_HF_DIV to LPI2C1.

enumerator kCLK_IN_to_LPI2C1
Attach CLK_IN to LPI2C1.

enumerator kCLK_1M_to_LPI2C1
Attach CLK_1M to LPI2C1.

enumerator kPll1ClkDiv_to_LPI2C1
Attach Pll1ClkDiv to LPI2C1.

enumerator kNONE_to_LPI2C1
Attach NONE to LPI2C1.

enumerator kFRO_LF_DIV_to_LPI2C2
Attach FRO_LF_DIV to LPI2C2.

enumerator kFRO_HF_DIV_to_LPI2C2
Attach FRO_HF_DIV to LPI2C2.

enumerator kCLK_IN_to_LPI2C2
Attach CLK_IN to LPI2C2.

enumerator kCLK_1M_to_LPI2C2
Attach CLK_1M to LPI2C2.

enumerator kPll1ClkDiv_to_LPI2C2
Attach Pll1ClkDiv to LPI2C2.

enumerator kNONE_to_LPI2C2
Attach NONE to LPI2C2.

enumerator kFRO_LF_DIV_to_LPI2C3
Attach FRO_LF_DIV to LPI2C3.

enumerator kFRO_HF_DIV_to_LPI2C3
Attach FRO_HF_DIV to LPI2C3.

enumerator kCLK_IN_to_LPI2C3
Attach CLK_IN to LPI2C3.

enumerator kCLK_1M_to_LPI2C3
Attach CLK_1M to LPI2C3.

enumerator kPll1ClkDiv_to_LPI2C3
Attach Pll1ClkDiv to LPI2C3.

enumerator kNONE_to_LPI2C3
Attach NONE to LPI2C3.

enumerator kFRO_LF_DIV_to_LPI2C4
Attach FRO_LF_DIV to LPI2C4.

enumerator kFRO_HF_DIV_to_LPI2C4
Attach FRO_HF_DIV to LPI2C4.

enumerator kCLK_IN_to_LPI2C4
Attach CLK_IN to LPI2C4.

enumerator kCLK_1M_to_LPI2C4
Attach CLK_1M to LPI2C4.

enumerator kPll1ClkDiv_to_LPI2C4
Attach Pll1ClkDiv to LPI2C4.

enumerator kNONE_to_LPI2C4
Attach NONE to LPI2C4.

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 kPll1ClkDiv_to_LPSPi0
Attach Pll1ClkDiv to LPSPi0.

enumerator kNONE_to_LPSPi0
Attach NONE to LPSPi0.

enumerator kFRO_LF_DIV_to_LPSPi1
Attach FRO_LF_DIV to LPSPi1.

enumerator kFRO_HF_DIV_to_LPSPi1
Attach FRO_HF_DIV to LPSPi1.

enumerator kCLK_IN_to_LPSPi1
Attach CLK_IN to LPSPi1.

enumerator kCLK_1M_to_LPSPi1
Attach CLK_1M to LPSPi1.

enumerator kPll1ClkDiv_to_LPSPi1
Attach Pll1ClkDiv to LPSPi1.

enumerator kNONE_to_LPSPi1
Attach NONE to LPSPi1.

enumerator kFRO_LF_DIV_to_LPSPi2
Attach FRO_LF_DIV to LPSPi2.

enumerator kFRO_HF_DIV_to_LPSPi2
Attach FRO_HF_DIV to LPSPi2.

enumerator kCLK_IN_to_LPSPi2
Attach CLK_IN to LPSPi2.

enumerator kCLK_1M_to_LPSPi2
Attach CLK_1M to LPSPi2.

enumerator kPll1ClkDiv_to_LPSPi2
Attach Pll1ClkDiv to LPSPi2.

enumerator kNONE_to_LPSPi2
Attach NONE to LPSPi2.

enumerator kFRO_LF_DIV_to_LPSPi3
Attach FRO_LF_DIV to LPSPi3.

enumerator kFRO_HF_DIV_to_LPSPi3
Attach FRO_HF_DIV to LPSPi3.

enumerator kCLK_IN_to_LPSPi3
Attach CLK_IN to LPSPi3.

enumerator kCLK_1M_to_LPSPi3
Attach CLK_1M to LPSPi3.

enumerator kPll1ClkDiv_to_LPSPi3
Attach Pll1ClkDiv to LPSPi3.

enumerator kNONE_to_LPSPi3
Attach NONE to LPSPi3.

enumerator kFRO_LF_DIV_to_LPSPi4
Attach FRO_LF_DIV to LPSPi4.

enumerator kFRO_HF_DIV_to_LPSPi4
Attach FRO_HF_DIV to LPSPi4.

enumerator kCLK_IN_to_LPSPi4
Attach CLK_IN to LPSPi4.

enumerator kCLK_1M_to_LPSPi4
Attach CLK_1M to LPSPi4.

enumerator kPll1ClkDiv_to_LPSPi4
Attach Pll1ClkDiv to LPSPi4.

enumerator kNONE_to_LPSPi4
Attach NONE to LPSPi4.

enumerator kFRO_LF_DIV_to_LPSPi5
Attach FRO_LF_DIV to LPSPi5.

enumerator kFRO_HF_DIV_to_LPSPi5
Attach FRO_HF_DIV to LPSPi5.

enumerator kCLK_IN_to_LPSPi5
Attach CLK_IN to LPSPi5.

enumerator kCLK_1M_to_LPSPi5
Attach CLK_1M to LPSPi5.

enumerator kPll1ClkDiv_to_LPSPi5
Attach Pll1ClkDiv to LPSPi5.

enumerator kNONE_to_LPSPi5
Attach NONE to LPSPi5.

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 kLP_OSC_to_LPUART0
Attach LP_OSC to LPUART0.

enumerator kCLK_1M_to_LPUART0
Attach CLK_1M to LPUART0.

enumerator kPll1ClkDiv_to_LPUART0
Attach Pll1ClkDiv 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 kLP_OSC_to_LPUART1
Attach LP_OSC to LPUART1.

enumerator kCLK_1M_to_LPUART1
Attach CLK_1M to LPUART1.

enumerator kPll1ClkDiv_to_LPUART1
Attach Pll1ClkDiv 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 kLP_OSC_to_LPUART2
Attach LP_OSC to LPUART2.

enumerator kCLK_1M_to_LPUART2
Attach CLK_1M to LPUART2.

enumerator kPll1ClkDiv_to_LPUART2
Attach Pll1ClkDiv to LPUART2.

enumerator kNONE_to_LPUART2
Attach NONE to LPUART2.

enumerator kFRO_LF_DIV_to_LPUART3
Attach FRO_LF_DIV to LPUART3.

enumerator kFRO_HF_DIV_to_LPUART3
Attach FRO_HF_DIV to LPUART3.

enumerator kCLK_IN_to_LPUART3
Attach CLK_IN to LPUART3.

enumerator kLP_OSC_to_LPUART3
Attach LP_OSC to LPUART3.

enumerator kCLK_1M_to_LPUART3
Attach CLK_1M to LPUART3.

enumerator kPll1ClkDiv_to_LPUART3
Attach Pll1ClkDiv to LPUART3.

enumerator kNONE_to_LPUART3
Attach NONE to LPUART3.

enumerator kFRO_LF_DIV_to_LPUART4
Attach FRO_LF_DIV to LPUART4.

enumerator kFRO_HF_DIV_to_LPUART4
Attach FRO_HF_DIV to LPUART4.

enumerator kCLK_IN_to_LPUART4
Attach CLK_IN to LPUART4.

enumerator kLP_OSC_to_LPUART4
Attach LP_OSC to LPUART4.

enumerator kCLK_1M_to_LPUART4
Attach CLK_1M to LPUART4.

enumerator kPll1ClkDiv_to_LPUART4
Attach Pll1ClkDiv to LPUART4.

enumerator kNONE_to_LPUART4
Attach NONE to LPUART4.

enumerator kFRO_LF_DIV_to_LPUART5
Attach FRO_LF_DIV to LPUART5.

enumerator kFRO_HF_DIV_to_LPUART5
Attach FRO_HF_DIV to LPUART5.

enumerator kCLK_IN_to_LPUART5
Attach CLK_IN to LPUART5.

enumerator kLP_OSC_to_LPUART5
Attach LP_OSC to LPUART5.

enumerator kCLK_1M_to_LPUART5
Attach CLK_1M to LPUART5.

enumerator kPll1ClkDiv_to_LPUART5
Attach Pll1ClkDiv to LPUART5.

enumerator kNONE_to_LPUART5
Attach NONE to LPUART5.

enumerator kCLK_32K_to_USBHS
Attach FRO_HF to USBHS.

enumerator kCLK_1M_to_USBHS
Attach CLK_IN to USBHS.

enumerator kPHY_CLK_XTAL_to_USBHS

Attach CLK_IN to USBHS.

enumerator kNONE_to_USBHS

Attach NONE to USBHS.

enumerator kFRO_HF_to_USBHS_PHY

Attach FRO_HF to USBHS_PHY.

enumerator kCLK_IN_to_USBHS_PHY

Attach CLK_IN to USBHS_PHY.

enumerator kNONE_to_USBHS_PHY

Attach NONE to USBHS_PHY.

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 kPll1ClkDiv_to_LPTMR0

Attach Pll1ClkDiv to LPTMR0.

enumerator kNONE_to_LPTMR0

Attach NONE to LPTMR0.

enumerator kCLK_16K_to_OSTIMER

Attach FRO16K to OSTIMER0.

enumerator kCLK_1M_to_OSTIMER

Attach CLK_1M to OSTIMER0.

enumerator kNONE_to_OSTIMER

Attach NONE to OSTIMER0.

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 kUSB_PLL_to_ADC

Attach USB_PL to ADC.

enumerator kCLK_1M_to_ADC

Attach CLK_1M to ADC.

enumerator kPll1ClkDiv_to_ADC

Attach Pll1ClkDiv 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 kPll1ClkDiv_to_CMP0
Attach Pll1ClkDiv 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 kLP_OSC_to_CLKOUT
Attach LP_OSC to CLKOUT.

enumerator kPll1ClkDiv_to_CLKOUT
Attach Pll1Clk 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 kCLK_IN_to_TENBASET_PHY
Attach CLK_IN to TENBASET_PHY.

enumerator kPll1Clk_to_TENBASET_PHY
Attach Pll1Clk to TENBASET_PHY.

enumerator kNONE_to_TENBASET_PHY
Attach NONE to TENBASET_PHY.

enumerator kFRO_HF_to_ESPI
Attach FRO_HF to ESPI.

enumerator kPll1ClkDiv_to_ESPI
Attach Pll1ClkDiv to ESPI.

enumerator kNONE_to_ESPI
Attach NONE to ESPI.

enumerator kFRO_HF_to_FLEXSPI
Attach FRO_HF to FLEXSPI.

enumerator kUSB_PFD_to_FLEXSPI
Attach USB_PFD to FLEXSPI.

enumerator kPll1Clk_to_FLEXSPI
Attach Pll1Clk to FLEXSPI.

enumerator kNONE_to_FLEXSPI
Attach NONE to FLEXSPI.

enumerator kFRO_LF_DIV_to_TSI0
Attach FRO_LF_DIV to TSI0.

enumerator kFRO_HF_DIV_to_TSI0
Attach FRO_HF_DIV to TSI0.

enumerator kCLK_IN_to_TSI0
Attach CLK_IN to TSI0.

enumerator kCLK_1M_to_TSI0
Attach CLK_1M to TSI0.

enumerator kPll1ClkDiv_to_TSI0
Attach Pll1ClkDiv to TSI0.

enumerator kNONE_to_TSI0
Attach NONE to TSI0.

enumerator kCLK_IN_to_ENETRMII
Attach CLK_IN to ENETRMII.

enumerator kPll1Clk_to_ENETRMII
Attach Pll1Clk to ENETRMII.

enumerator kNONE_to_ENETRMII

Attach NONE to ENETRMII.

enumerator kCLK_IN_to_ENETPTPREF

Attach LK_IN to ENETPTPREF.

enumerator kENET0_TX_CLK_to_ENETPTPREF

Attach ENET0_TX_CLK to ENETPTPREF.

enumerator kPll1Clk_to_ENETPTPREF

Attach Pll1Clk to ENETRMII.

enumerator kNONE_to_ENETPTPREF

Attach NONE to ENETPTPREF.

enumerator kNONE_to_NONE

Attach NONE to NONE.

enum _clock_div_name

Clock dividers.

Values:

enumerator kCLOCK_DivI3C0_FCLK

I3C0_FCLK clock divider

enumerator kCLOCK_DivI3C1_FCLK

I3C1_FCLK clock divider

enumerator kCLOCK_DivCTIMER0

CTIMER0 clock divider

enumerator kCLOCK_DivCTIMER1

CTIMER1 clock divider

enumerator kCLOCK_DivCTIMER2

CTIMER2 clock divider

enumerator kCLOCK_DivCTIMER3

CTIMER3 clock divider

enumerator kCLOCK_DivCTIMER4

CTIMER4 clock divider

enumerator kCLOCK_DivWWDTO

WWDTO clock divider

enumerator kCLOCK_DivWWDT1

WWDT1 clock divider

enumerator kCLOCK_DivE1588

E1588 clock divider

enumerator kCLOCK_DivRMII

RMII clock divider

enumerator kCLOCK_DivESPI0

ESPI0 clock divider

enumerator kCLOCK_DivFLEXSPI0

FLEXSPI0 clock divider

enumerator kCLOCK_DivLPSPI2
LPSPI2 clock divider

enumerator kCLOCK_DivLPSPI3
LPSPI3 clock divider

enumerator kCLOCK_DivLPSPI4
LPSPI4 clock divider

enumerator kCLOCK_DivLPSPI5
LPSPI5 clock divider

enumerator kCLOCK_DivTENBASET_PHY0
TENBASET_PHY0 clock divider

enumerator kCLOCK_DivUSBHS_PHY
USBHS_PHY clock divider

enumerator kCLOCK_DivFLEXIO0
FLEXIO0 clock divider

enumerator kCLOCK_DivLPI2C0
LPI2C0 clock divider

enumerator kCLOCK_DivLPI2C1
LPI2C1 clock divider

enumerator kCLOCK_DivLPSPI0
LPSPI0 clock divider

enumerator kCLOCK_DivLPSPI1
LPSPI1 clock divider

enumerator kCLOCK_DivI3C2_FCLK
I3C2_FCLK 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_DivLPUART4
LPUART4 clock divider

enumerator kCLOCK_DivLPTMR0
LPTMR0 clock divider

enumerator kCLOCK_DivADC
ADCx clock divider

enumerator kCLOCK_DivCMP0_FUNC
CMP0_FUNC clock divider

enumerator kCLOCK_DivCMP0_RR
CMP0_RR clock divider

enumerator kCLOCK_DivDAC0
DAC0 clock divider

enumerator kCLOCK_DivDAC1
DAC1 clock divider

enumerator kCLOCK_DivTSI0
TSI0 clock divider

enumerator kCLOCK_DivFLEXCAN0
FLEXCAN0 clock divider

enumerator kCLOCK_DivFLEXCAN1
FLEXCAN1 clock divider

enumerator kCLOCK_DivLPI2C2
LPI2C2 clock divider

enumerator kCLOCK_DivLPI2C3
LPI2C3 clock divider

enumerator kCLOCK_DivLPI2C4
LPI2C4 clock divider

enumerator kCLOCK_DivLPUART5
LPUART5 clock divider

enumerator kCLOCK_DivI3C3_FCLK
I3C3_FCLK clock divider

enumerator kCLOCK_DivTRACE
TRACE clock divider

enumerator kCLOCK_DivCLKOUT
CLKOUT clock divider

enumerator kCLOCK_DivSYSTICK
SYSTICK clock divider

enumerator kCLOCK_DivSLOWCLK
SLOWCLK clock divider

enumerator kCLOCK_DivBUSCLK
BUSCLK clock divider

enumerator kCLOCK_DivAHBCLK
AHBCLK clock divider

enumerator kCLOCK_DivFRO_HF
FROHF clock divider

enumerator kCLOCK_DivFRO_LF
FROLF clock divider

enumerator kCLOCK_DivPLL1CLK
PLL1CLK clock divider

enum _clke_16k
FRO16K connection to each power domain.

Values:

enumerator kCLKE_16K_SYSTEM

To VSYS domain.

enumerator kCLKE_16K_COREMAIN

To VDD_CORE domain.

enumerator kCLKE_16K_VBAT

To VBAT domain.

enumerator kCLKE_16K_ALL

To VSYS,VDD_CORE,VBAT 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 _scg_pll1_monitor_mode

SCG PLL1 monitor mode.

Values:

enumerator kSCG_Pll1MonitorDisable
Monitor disabled.

enumerator kSCG_Pll1MonitorInt
Interrupt when the PLL1 Clock error is detected.

enumerator kSCG_Pll1MonitorReset
Reset when the PLL1 Clock error is detected.

enum __run__mode
The active run mode (voltage level).
Values:

enumerator kMD_Mode
Middle driver mode, VDD_CORE: 1.0V

enumerator kSD_Mode
Standard driver mode,VDD_CORE: 1.1V

enumerator kOD_Mode
Over drive mode, VDD_CORE: 1.2V

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_Osc32kToVbat
OSC32K[2] to VBAT domain.

enumerator kCLOCK_Osc32kToAll
OSC32K to SYSTEM,CORE,VBAT 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

enum _pfd_clkout_selection_t

Source of PFD Clock Selection.

Values:

enumerator kPFDOUT__UsbPllRef

enumerator kPFDOUT__PfdClkDiv4

enumerator kPFDOUT__PfdClkDiv2

enumerator kPFDOUT__PfdClk

enum _pll_clk_src

PLL clock source.

Values:

enumerator kPll__ClkSrcSysOsc

System OSC.

enumerator kPll__ClkSrcFirc

Fast IRC.

enumerator kPll__ClkSrcSirc

Slow IRC.

enum _ss_progmodfm

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

Values:

enumerator kSS__MF_512

Nss = 512 (fm ~= 3.9 - 7.8 kHz)

enumerator kSS__MF_384

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

enumerator kSS__MF_256

Nss = 256 (fm ~= 7.8 - 15.6 kHz)

enumerator kSS__MF_128

Nss = 128 (fm ~= 15.6 - 31.3 kHz)

enumerator kSS__MF_64

Nss = 64 (fm ~= 32.3 - 64.5 kHz)

enumerator kSS__MF_32

Nss = 32 (fm ~= 62.5 - 125 kHz)

enumerator kSS__MF_24

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

enumerator kSS_MF_16

Nss = 16 (fm ~= 125 - 250 kHz)

enum _ss_progmoddp

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

Values:

enumerator kSS_MR_K0

k = 0 (no spread spectrum)

enumerator kSS_MR_K1

k ~= 1

enumerator kSS_MR_K1_5

k ~= 1.5

enumerator kSS_MR_K2

k ~= 2

enumerator kSS_MR_K3

k ~= 3

enumerator kSS_MR_K4

k ~= 4

enumerator kSS_MR_K6

k ~= 6

enumerator kSS_MR_K8

k ~= 8

enum _ss_modwvctrl

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

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

Values:

enumerator kSS_MC_NOC

no compensation

enumerator kSS_MC_RECC

recommended setting

enumerator kSS_MC_MAXC

max. compensation

enum _pll_error

PLL status definitions.

Values:

enumerator kStatus_PLL_Success

PLL operation was successful

enumerator kStatus_PLL_OutputTooLow

PLL output rate request was too low

enumerator `kStatus_PLL_OutputTooHigh`
PLL output rate request was too high

enumerator `kStatus_PLL_OutputError`
PLL output rate error

enumerator `kStatus_PLL_InputTooLow`
PLL input rate is too low

enumerator `kStatus_PLL_InputTooHigh`
PLL input rate is too high

enumerator `kStatus_PLL_OutsideIntLimit`
Requested output rate isn't possible

enumerator `kStatus_PLL_CCOTooLow`
Requested CCO rate isn't possible

enumerator `kStatus_PLL_CCOTooHigh`
Requested CCO rate isn't possible

typedef enum `_clock_ip_name` `clock_ip_name_t`
Clock gate name used for `CLOCK_EnableClock/CLOCK_DisableClock`.

typedef enum `_clock_name` `clock_name_t`
Clock name used to get clock frequency.

typedef enum `_clock_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.
MAX clock divider

typedef enum `_clke_16k` `clke_16k_t`
FRO16K connection to each power domain.

typedef enum `_sirc_trim_mode` `sirc_trim_mode_t`
sirc trim mode.

typedef enum `_sirc_trim_src` `sirc_trim_src_t`
sirc trim source.

typedef struct `_sirc_trim_config` `sirc_trim_config_t`
sirc trim configuration.

typedef enum `_scg_sosc_monitor_mode` `scg_sosc_monitor_mode_t`
SCG system OSC monitor mode.

typedef enum `_scg_pll1_monitor_mode` `scg_pll1_monitor_mode_t`
SCG PLL1 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.

`typedef enum _pfd_clkout_selection_t pfd_clkout_selection_t`

Source of PFD Clock Selection.

`typedef enum _pll_clk_src pll_clk_src_t`

PLL clock source.

`typedef enum _ss_progmodfm ss_progmodfm_t`

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

`typedef enum _ss_progmoddp ss_progmoddp_t`

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

`typedef enum _ss_modwvctrl ss_modwvctrl_t`

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

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

`typedef struct _pll_config pll_config_t`

PLL configuration structure.

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

```
typedef struct _pll_setup pll_setup_t
```

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

```
typedef enum _pll_error pll_error_t
```

PLL status definitions.

```
static inline void CLOCK_EnableClock(clock_ip_name_t clk)
```

Enable the clock for specific IP.

Parameters

- *clk* – : Clock to be enabled.

Returns

Nothing

```
static inline void CLOCK_DisableClock(clock_ip_name_t clk)
```

Disable the clock for specific IP.

Parameters

- *clk* – : Clock to be Disabled.

Returns

Nothing

```
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` – 0-7 0: disable, 1: disable bit0 is for `clk_16k0` bit1 is for `clk_16k1` bit2 is for `clk_16k2`

Returns

returns success or fail status.

`status_t CLOCK_SetupExtClocking(uint32_t iFreq)`

Initialize the external osc clock to given frequency.

Parameters

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

Returns

returns success or fail status.

`status_t CLOCK_SetupExtRefClocking(uint32_t iFreq)`

Initialize the external reference clock to given frequency.

Parameters

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

Returns

returns success or fail status.

status_t CLOCK_SetupOsc32KClocking(*uint32_t* id)

Initialize the XTAL32/EXTAL32 input clock to given frequency.

Parameters

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

Returns

returns success or fail status.

void CLOCK_GetDefaultOsc32KConfig(*osc_32k_config_t* *config)

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

Parameters

- config – Pointer to a configuration structure

status_t CLOCK_SetupOsc32KClockingConfig(*osc_32k_config_t* config)

Initialize the OSC 32K with user-defined settings.

Parameters

- config – : OSC 32K configuration structure

Returns

returns success or fail status.

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_GetI3CFClkFreq(uint32_t id)
Return Frequency of I3C Clock.

Returns

Frequency of I3C Clock

uint32_t CLOCK_GetLptmrClkFreq(void)
Return Frequency of LPTMR functional Clock.

Returns

Frequency of LPTMR functional Clock

uint32_t CLOCK_GetOstimerClkFreq(void)
Return Frequency of OSTIMER.

Returns

Frequency of OSTIMER Clock

uint32_t CLOCK_GetAdcClkFreq(uint32_t id)
Return Frequency of Adc Clock.

Returns

Frequency of Adc.

uint32_t CLOCK_GetCmpFClkFreq(void)
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_GetWwdt0ClkFreq(void)
brief Return Frequency of WWDTO Clock return Frequency of WWDTO.

uint32_t CLOCK_GetWwdt1ClkFreq(void)
brief Return Frequency of WWDT1 Clock return Frequency of WWDT1.

uint32_t CLOCK_GetFlexcanClkFreq(uint32_t id)

Return Frequency of FlexCAN CLK.

Returns

Frequency of FlexCAN CLK.

uint32_t CLOCK_GetFlexspiClkFreq(void)

Return Frequency of Flexspi CLK.

Returns

Frequency of Flexspi CLK.

uint32_t CLOCK_GetTsiClkFreq(void)

Return Frequency of TSI CLK.

Returns

Frequency of TSI CLK.

uint32_t CLOCK_GetEnetRmiiClkFreq(void)

Return Frequency of ENET MRR CLK.

Returns

Frequency of ENET MRR CLK.

uint32_t CLOCK_GetEnetPtpRefClkFreq(void)

Return Frequency of ENET PTP REF CLK.

Returns

Frequency of ENET PTP REF CLK.

uint32_t CLOCK_GetTenbaset_PhyClkFreq(void)

Return Frequency of TENBASET_PHY CLK.

Returns

Frequency of TENBASET_PHY CLK.

uint32_t CLOCK_GetEspiclKFreq(void)

Return Frequency of ESPI CLK.

Returns

Frequency of ESPI CLK.

uint32_t CLOCK_GetFlexioClkFreq(void)

Return Frequency of FLEXIO FCLK.

Returns

Frequency of FLEXIO FCLK.

void CLOCK_SetupEnetTxClk(uint32_t iFreq)

Initialize the ENET TX CLK to given frequency.

Parameters

- iFreq – : Desired frequency

Returns

Nothing

uint32_t CLOCK_GetEnetTxClkFreq(void)

Return Frequency of ENET TX CLK.

Returns

Frequency of ENET TX CLK

`bool CLOCK_EnableUsbhsPhyPllClock(uint32_t clockSourceFreq)`

Enable USB HS PHY PLL clock.

This function enables the USB HS PHY PLL clock.

Parameters

- `clockSourceFreq` – Frequency value.

Returns

true if success, false if failure.

`void CLOCK_DisableUsbhsPhyPllClock(void)`

Disable USB HS PHY PLL clock.

This function disables the USB HS PHY PLL clock.

`bool CLOCK_EnableUsbhsClock(void)`

Enable USB HS clock.

This function enables the USB HS clock.

Returns

true if success, false if failure.

`bool CLOCK_EnableUsbhsPhyPfdClock(uint32_t pfdDiv, pfd_clkout_selection_t pfdClkSel)`

Enable USB HS PHY PFD clock.

This function enables the USB HS PHY PFD clock.

Parameters

- `pfdDiv` – PFD fractional divider value.
- `pfdClkSel` – PFD clock output selection.

Returns

true if success, false if failure.

`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

`void CLOCK_SetPLL1MonitorMode(scg_pll1_monitor_mode_t mode)`

Sets the PLL1 monitor mode.

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

Parameters

- mode – Monitor mode to set.

`uint32_t CLOCK_GetPLL1InClockRate(void)`

Return PLL1 input clock rate.

Returns

PLL1 input clock rate

`__STATIC_INLINE bool CLOCK_IsPLL1Locked (void)`

Check if PLL1 is locked or not.

Returns

true if the PLL1 is locked, false if not locked

`uint32_t CLOCK_GetPLLOutFromSetup(pll_setup_t *pSetup)`

Return PLL0 output clock rate from setup structure.

Parameters

- pSetup – : Pointer to a PLL setup structure

Returns

System PLL output clock rate the setup structure will generate

`pll_error_t CLOCK_SetupPLLDData(pll_config_t *pControl, pll_setup_t *pSetup)`

Set PLL output based on the passed PLL setup data.

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

Parameters

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

Returns

PLL_ERROR_SUCCESS on success, or PLL setup error code

`pll_error_t CLOCK_SetPLL1Freq(const pll_setup_t *pSetup)`

Set PLL output from PLL setup structure (precise frequency)

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

Parameters

- pSetup – : Pointer to populated PLL setup structure

Returns

kStatus_PLL_Success on success, or PLL setup error code

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.

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)

kCLOCK_Crc0

kCLOCK_Ewm0

kCLOCK_InputMux

kCLOCK_Smartdma

AOI_CLOCKS

Clock ip name array for AOI.

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.

ESPI_CLOCKS

Clock gate name array for ESPI.

ETH_CLOCKS

Clock gate name array for ENET.

E1588_CLOCKS

Clock gate name array for E1588.

FLEXCAN_CLOCKS

Clock ip name array for FLEXCAN.

FREQME_CLOCKS

 Clock ip name array for FREQME.

FLEXIO_CLOCKS

 Clock ip name array for FLEXIO.

FLEXSPI_CLOCKS

 Clock ip name array for FLEXSPI.

GPIO_CLOCKS

 Clock ip name array for GPIO.

GDET_CLOCKS

 Clock ip name array for GDET.

INPUTMUX_CLOCKS

 Clock ip name array for INPUTMUX.

LPCMP_CLOCKS

 Clock ip name array for GPIO.

LPADC_CLOCKS

 Clock ip name array for LPADC.

LPDAC_CLOCKS

 Clock ip name array for LPDAC.

LPUART_CLOCKS

 Clock ip name array for LPUART.

LPI2C_CLOCKS

 Clock ip name array for LPI2C.

LPSPIC_CLOCKS

 Clock ip name array for LSPI.

I3C_CLOCKS

 Clock ip name array for I3C.

OSTIMER_CLOCKS

 Clock ip name array for OSTIMER.

PORT_CLOCKS

 Clock ip name array for PORT.

RMII_CLOCKS

 Clock gate name array for RMII.

SPIFILTER_CLOCKS

 Clock ip name array for SPIFILTER.

SMARTDMA_CLOCKS

 Clock ip name array for SMARTDMA.

TRNG_CLOCKS

 Clock ip name array for TRNG.

UTICK_CLOCKS

 Clock ip name array for UTICK.

USB_CLOCKS

 Clock ip name array for USBHS.

USBPHY_CLOCKS

Clock ip name array for USBHSPHY.

VREF_CLOCKS

Clock ip name array for VREF.

RTC_CLOCKS

Clock gate name array for RTC.

TSI_CLOCKS

Clock gate name array for TSI.

TENBASET_PHY_CLOCKS

Clock gate name array for TENBASET_PHY.

WWDT_CLOCKS

Clock ip name array for WWDT.

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)

kCLOCK_DivMax

PLL_CONFIGFLAG_FORCENOFRACT

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

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

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

sirc_trim_mode_t trimMode

Trim mode.

sirc_trim_src_t trimSrc

Trim source.

uint16_t trimDiv

Divider of SOSC.

uint8_t cltrim

Trim coarse value; Irrelevant if trimMode is kSCG_TrimUpdate.

uint8_t ccotrim

Trim fine value; Irrelevant if trimMode is kSCG_TrimUpdate.

bool enableInternalCapBank

enable/disable the internal capacitance bank.

`bool enableCrystalOscillatorBypass`

enable/disable the crystal oscillator bypass.

`vbat_osc_xtal_cap_t xtalCap`

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

`vbat_osc_extal_cap_t extalCap`

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

`vbat_osc_coarse_adjustment_value_t coarseAdjustment`

32kHz crystal oscillator amplifier coarse adjustment value.

`vbat_osc_init_trim_t initTrim`

`vbat_osc_cap_trim_t capTrim`

`vbat_osc_dly_trim_t dlyTrim`

`vbat_osc_cap2_trim_t cap2Trim`

`vbat_osc_cmp_trim_t cmpTrim`

`vbat_osc_mode_en_t mode`

`vbat_osc_xtal_cap_t xtalCap`

`vbat_osc_extal_cap_t extalCap`

`vbat_osc_coarse_adjustment_value_t ampGain`

`osc32k_clk_gate_id_t id`

`uint32_t desiredRate`

Desired PLL rate in Hz

`uint32_t inputSource`

PLL input source

`uint32_t flags`

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

`ss_progmodfm_t ss_mf`

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

`ss_progmoddp_t ss_mr`

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

`ss_modwvctrl_t ss_mc`

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

`bool mfDither`

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

`uint32_t pllctrl`

PLL Control register APLLCTRL

uint32_t pllndiv

PLL N Divider register APLLNDIV

uint32_t pllpdiv

PLL P Divider register APLLPDIV

uint32_t pllmdiv

PLL M Divider register APLLMDIV

uint32_t pllsscg[2]

PLL Spread Spectrum Control registers APLLSSCG

uint32_t pllRate

Actual PLL rate

struct __sirc_trim_config

#include <fsl_clock.h> sirc trim configuration.

struct __vbat_osc_config

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

struct __osc_32k_config

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

struct __pll_config

#include <fsl_clock.h> PLL configuration structure.

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

struct __pll_setup

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

2.4 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.1.0.

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()

2.5 CTIMER: Standard counter/timers

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

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

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

Parameters

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

void CTIMER_Deinit(CTIMER_Type *base)

Gates the timer clock.

Parameters

- base – Ctimer peripheral base address

void CTIMER_GetDefaultConfig(*ctimer_config_t* *config)

Fills in the timers configuration structure with the default settings.

The default values are:

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

Parameters

- config – Pointer to the user configuration structure.

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

Configures the PWM signal parameters.

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

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

Parameters

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

Returns

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

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

Configures the PWM signal parameters.

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

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

Parameters

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

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

Updates the pulse period of an active PWM signal.

Parameters

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

```
status_t CTIMER_UpdatePwmDutycycle(CTIMER_Type *base, const ctimer_match_t
                                   pwmPeriodChannel, ctimer_match_t matchChannel,
                                   uint8_t dutyCyclePercent)
```

Updates the duty cycle of an active PWM signal.

Note: Please use CTIMER_SetupPwmPeriod to update the PWM with high resolution. This function can manually assign the specified channel to set the PWM cycle.

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- dutyCyclePercent – New PWM pulse width; the value should be between 0 to 100

Returns

kStatus_Success on success kStatus_Fail If PWM pulse width register value is larger than 0xFFFFFFFF.

```
static inline void CTIMER_EnableInterrupts(CTIMER_Type *base, uint32_t mask)
```

Enables the selected Timer interrupts.

Parameters

- base – Ctimer peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ctimer_interrupt_enable_t*

```
static inline void CTIMER_DisableInterrupts(CTIMER_Type *base, uint32_t mask)
```

Disables the selected Timer interrupts.

Parameters

- base – Ctimer peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ctimer_interrupt_enable_t*

```
static inline uint32_t CTIMER_GetEnabledInterrupts(CTIMER_Type *base)
```

Gets the enabled Timer interrupts.

Parameters

- base – Ctimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *ctimer_interrupt_enable_t*

static inline uint32_t CTIMER_GetStatusFlags(CTIMER_Type *base)

Gets the Timer status flags.

Parameters

- base – Ctimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration `ctimer_status_flags_t`

static inline void CTIMER_ClearStatusFlags(CTIMER_Type *base, uint32_t mask)

Clears the Timer status flags.

Parameters

- base – Ctimer peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration `ctimer_status_flags_t`

static inline void CTIMER_StartTimer(CTIMER_Type *base)

Starts the Timer counter.

Parameters

- base – Ctimer peripheral base address

static inline void CTIMER_StopTimer(CTIMER_Type *base)

Stops the Timer counter.

Parameters

- base – Ctimer peripheral base address

FSL_CTIMER_DRIVER_VERSION

Version 2.3.4

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], kCTIMER_MultipleCallback);

Parameters

- base – Ctimer peripheral base address
- cb_func – Pointer to callback function pointer
- cb_type – callback function type, singular or multiple

static inline void CTIMER_Reset(CTIMER_Type *base)

Reset the counter.

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

- base – Ctimer peripheral base address

static inline void CTIMER_SetPrescale(CTIMER_Type *base, uint32_t prescale)

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

Parameters

- base – Ctimer peripheral base address
- prescale – Prescale value

static inline uint32_t CTIMER_GetCaptureValue(CTIMER_Type *base, *ctimer_capture_channel_t*
capture)

Get capture channel value.

Get the counter/timer value on the corresponding capture channel.

Parameters

- base – Ctimer peripheral base address

- capture – Select capture channel

Returns

The timer count capture value.

```
static inline void CTIMER__EnableResetMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                    match, bool enable)
```

Enable reset match channel.

Set the specified match channel reset operation.

Parameters

- base – Ctimer peripheral base address
- match – match channel used
- enable – Enable match channel reset operation.

```
static inline void CTIMER__EnableStopMatchChannel(CTIMER_Type *base, ctimer_match_t
                                                    match, bool enable)
```

Enable stop match channel.

Set the specified match channel stop operation.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable match channel stop operation.

```
static inline void CTIMER__EnableMatchChannelReload(CTIMER_Type *base, ctimer_match_t
                                                    match, bool enable)
```

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable .

```
static inline void CTIMER__EnableRisingEdgeCapture(CTIMER_Type *base,
                                                    ctimer_capture_channel_t capture, bool
                                                    enable)
```

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable rising edge capture.

```
static inline void CTIMER__EnableFallingEdgeCapture(CTIMER_Type *base,
                                                    ctimer_capture_channel_t capture, bool
                                                    enable)
```

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable falling edge capture.

```
static inline void CTIMER_SetShadowValue(CTIMER_Type *base, ctimer_match_t match,
                                         uint32_t matchvalue)
```

Set the specified match shadow channel.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- matchvalue – Reload the value of the corresponding match register.

```
struct __ctimer_match_config
```

#include <fsl_ctimer.h> Match configuration.

This structure holds the configuration settings for each match register.

Public Members

uint32_t matchValue

This is stored in the match register

bool enableCounterReset

true: Match will reset the counter false: Match will not reset the counter

bool enableCounterStop

true: Match will stop the counter false: Match will not stop the counter

ctimer_match_output_control_t outControl

Action to be taken on a match on the EM bit/output

bool outPinInitState

Initial value of the EM bit/output

bool enableInterrupt

true: Generate interrupt upon match false: Do not generate interrupt on match

```
struct __ctimer_config
```

#include <fsl_ctimer.h> Timer configuration structure.

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the CTIMER_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

ctimer_timer_mode_t mode

Timer mode

ctimer_capture_channel_t input

Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC

uint32_t prescale

Prescale value

2.6 DAC: Digital-to-Analog Converter Driver

`void DAC_Init(LPDAC_Type *base, const dac_config_t *config)`

Initialize the DAC module with common configuration.

The clock will be enabled in this function.

Parameters

- `base` – DAC peripheral base address.
- `config` – Pointer to configuration structure.

`void DAC_GetDefaultConfig(dac_config_t *config)`

Get the default settings for initialization's configuration.

This function initializes the user configuration structure to a default value. The default values are:

```
config->fifoWatermarkLevel = 0U;
config->fifoTriggerMode = kDAC_FIFOTriggerByHardwareMode;
config->fifoWorkMode = kDAC_FIFODisabled;
config->enableLowPowerMode = false;
config->referenceVoltageSource = kDAC_ReferenceVoltageSourceAlt1;
```

Parameters

- `config` – Pointer to configuration structure.

`void DAC_Deinit(LPDAC_Type *base)`

De-initialize the DAC module.

The clock will be disabled in this function.

Parameters

- `base` – DAC peripheral base address.

`static inline void DAC_SetReset(LPDAC_Type *base, uint32_t mask)`

Assert the reset control to part hardware.

This function is to assert the reset control to part hardware. Responding part hardware would remain reset until cleared by software.

Parameters

- `base` – DAC peripheral base address.
- `mask` – The reset control mask, see to `_dac_reset_control_t`.

`static inline void DAC_ClearReset(LPDAC_Type *base, uint32_t mask)`

Clear the reset control to part hardware.

This function is to clear the reset control to part hardware. Responding part hardware would work after the reset control is cleared by software.

Parameters

- `base` – DAC peripheral base address.
- `mask` – The reset control mask, see to `_dac_reset_control_t`.

`static inline void DAC_Enable(LPDAC_Type *base, bool enable)`

Enable the DAC hardware system or not.

This function is to start the Programmable Reference Generator operation or not.

Parameters

- base – DAC peripheral base address.
- enable – Assertion of indicated event.

static inline void DAC_EnableInterrupts(LPDAC_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – DAC peripheral base address.
- mask – Mask value of indicated interrupt events. See to `_dac_interrupt_enable`.

static inline void DAC_DisableInterrupts(LPDAC_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – DAC peripheral base address.
- mask – Mask value of indicated interrupt events. See to `_dac_interrupt_enable`.

static inline void DAC_EnableDMA(LPDAC_Type *base, uint32_t mask, bool enable)

Enable the DMA switchers or not.

Parameters

- base – DAC peripheral base address.
- mask – Mask value of indicated DMA request. See to `_dac_dma_enable`.
- enable – Enable the DMA or not.

static inline uint32_t DAC_GetStatusFlags(LPDAC_Type *base)

Get status flags of DAC module.

Parameters

- base – DAC peripheral base address.

Returns

Mask value of status flags. See to `_dac_status_flags`.

static inline void DAC_ClearStatusFlags(LPDAC_Type *base, uint32_t flags)

Clear status flags of DAC module.

Parameters

- base – DAC peripheral base address.
- flags – Mask value of status flags to be cleared. See to `_dac_status_flags`.

static inline void DAC_SetData(LPDAC_Type *base, uint32_t value)

Set data into the entry of FIFO buffer.

Parameters

- base – DAC peripheral base address.
- value – Setting value into FIFO buffer.

static inline uint32_t DAC_GetFIFOWritePointer(LPDAC_Type *base)

Get the value of the FIFO write pointer.

Parameters

- base – DAC peripheral base address.

Returns

Current value of the FIFO write pointer.

static inline uint32_t DAC_GetFIFOReadPointer(LPDAC_Type *base)

Get the value of the FIFO read pointer.

Parameters

- base – DAC peripheral base address.

Returns

Current value of the FIFO read pointer.

static inline void DAC_DoSoftwareTriggerFIFO(LPDAC_Type *base)

Do software trigger to FIFO when in software mode.

Parameters

- base – DAC peripheral base address.

FSL_DAC_DRIVER_VERSION

DAC driver version 2.1.3.

DAC reset control.

Values:

enumerator kDAC_ResetFIFO

Resets the FIFO pointers and flags.

enumerator kDAC_ResetLogic

Resets all DAC registers and internal logic.

DAC interrupts.

Values:

enumerator kDAC_FIFOFullInterruptEnable

FIFO full interrupt enable.

enumerator kDAC_FIFOEmptyInterruptEnable

FIFO empty interrupt enable.

enumerator kDAC_FIFOWatermarkInterruptEnable

FIFO watermark interrupt enable.

enumerator kDAC_SwingBackInterruptEnable

Swing back one cycle complete interrupt enable.

enumerator kDAC_FIFOOverflowInterruptEnable

FIFO overflow interrupt enable.

enumerator kDAC_FIFOUnderflowInterruptEnable

FIFO underflow interrupt enable.

enumerator kDAC_PeriodTriggerCompleteInterruptEnable

Period trigger mode conversion complete interrupt enable

DAC DMA switchers.

Values:

enumerator kDAC_FIFOEmptyDMAEnable

FIFO empty DMA enable.

enumerator `kDAC_FIFOWatermarkDMAEnable`
FIFO watermark DMA enable.

DAC status flags.

Values:

enumerator `kDAC_FIFOUnderflowFlag`

This flag means that there is a new trigger after the buffer is empty. The FIFO read pointer will not increase in this case and the data sent to DAC analog conversion will not changed. This flag is cleared by writing a 1 to it.

enumerator `kDAC_FIFOverflowFlag`

This flag indicates that data is intended to write into FIFO after the buffer is full. The writer pointer will not increase in this case. The extra data will not be written into the FIFO. This flag is cleared by writing a 1 to it.

enumerator `kDAC_FIFOSwingBackFlag`

This flag indicates that the DAC has completed one period of conversion in swing back mode. It means that the read pointer has increased to the top (write pointer) once and then decreased to zero once. For example, after three data is written to FIFO, the writer pointer is now 3. Then, if continually triggered, the read pointer will swing like: 0-1-2-1-0-1-2-, and so on. After the fourth trigger, the flag is set. This flag is cleared by writing a 1 to it.

enumerator `kDAC_FIFOWatermarkFlag`

This field is set if the remaining data in FIFO is less than or equal to the setting value of watermark. By writing data into FIFO by DMA or CPU, this flag is cleared automatically when the data in FIFO is more than the setting value of watermark.

enumerator `kDAC_FIFOEmptyFlag`

FIFO empty flag.

enumerator `kDAC_FIFOFullFlag`

FIFO full flag.

enumerator `kDAC_PeriodTriggerCompleteFlag`

Period trigger mode conversion complete flag.

enum `_dac_fifo_trigger_mode`

DAC FIFO trigger mode.

Values:

enumerator `kDAC_FIFOTriggerByHardwareMode`

Buffer would be triggered by hardware.

enumerator `kDAC_FIFOTriggerBySoftwareMode`

Buffer would be triggered by software.

enum `_dac_fifo_work_mode`

DAC FIFO work mode.

Values:

enumerator `kDAC_FIFODisabled`

FIFO mode is disabled and buffer mode is enabled. Any data written to DATA[DATA] goes to buffer then goes to conversion.

enumerator `kDAC_FIFOWorkAsNormalMode`

FIFO mode is enabled. Data will be first read from FIFO to buffer then goes to conversion.

enumerator kDAC_FIFOWorkAsSwingMode

In swing mode, the read pointer swings between the writer pointer and zero. That is, the trigger increases the read pointer till reach the writer pointer and decreases the read pointer till zero, and so on. The FIFO empty/full/watermark flag will not update during swing back mode.

enumerator kDAC_FIFOWorkAsPeriodTriggerMode

In periodic trigger mode, user only needs to send the first trigger. Then after every [PTG_PERIOD+1] RCLK cycles, DAC will be automatically triggered by internal trigger. There will be [PTG_NUM] internal triggers, thus in total [PTG_NUM+1] conversions including the first trigger sent by user. User can terminate the current conversion queue by clearing the GCR[PTGEN] bit. Then, after the current conversion is completed, the conversion is terminated and the PTGCOCO flag is set. If PCR[PTG_NUM] is set to zero, there will be infinite triggers following the first hardware/software trigger, until the GCR[PTGEN] is cleared by software. In any case, the conversion can be terminated by FIFORST/SWRST.

enumerator kDAC_FIFOWorkAsPeriodTriggerAndSwingMode

Periodically trigger DAC and swing back.

enum _dac_reference_voltage_source

DAC reference voltage source.

Values:

enumerator kDAC_ReferenceVoltageSourceAlt1

The DAC selects VREFH_INT as the reference voltage.

enumerator kDAC_ReferenceVoltageSourceAlt2

The DAC selects VREFH_EXT as the reference voltage.

enum _dac_reference_current_source

Values:

enumerator kDAC_ReferenceCurrentSourcePtat

enumerator kDAC_ReferenceCurrentSourceZtc

typedef enum _dac_fifo_trigger_mode dac_fifo_trigger_mode_t

DAC FIFO trigger mode.

typedef enum _dac_fifo_work_mode dac_fifo_work_mode_t

DAC FIFO work mode.

typedef enum _dac_reference_voltage_source dac_reference_voltage_source_t

DAC reference voltage source.

typedef enum _dac_reference_current_source dac_reference_current_source_t

typedef struct _dac_config dac_config_t

DAC configuration structure.

struct _dac_config

#include <fsl_dac.h> DAC configuration structure.

Public Members

uint32_t fifoWatermarkLevel

FIFO's watermark, the max value can be the hardware FIFO size.

dac_fifo_trigger_mode_t fifoTriggerMode

Select the trigger mode for FIFO.

dac_fifo_work_mode_t fifoWorkMode

Select the work mode for FIFO.

bool enableOpampBuffer

Opamp is used as buffer.

bool enableLowerLowPowerMode

Enable the lower low power mode.

uint32_t periodicTriggerNumber

There will be 'periodicTriggerNumber' internal triggers following the first hardware/software trigger. So there will be 'periodicTriggerNumber + 1' conversions in total. If set to zero, there will be infinite triggers following the first hw/sw trigger, until the GCR[PTGEN] is cleared.

uint32_t periodicTriggerWidth

Control the periodic trigger frequency. There will be 'periodicTriggerWidth + 1' RCLK cycles between each periodic trigger. The periodic trigger frequency should be configured to not larger than the analog conversion speed.

uint32_t syncTime

RCLK cycles before data latch. accessible range is 0-15. It is used to configure the DAC sync cycles which is helpful to reduce glitch on the output. The sync time is (LATCH_CYC+1) RCLK cycles. User should configure this register according to the RCLK frequency. The recommended sync time is at least 40ns.

dac_reference_current_source_t referenceCurrentSource

Select the internal reference current source.

dac_reference_voltage_source_t referenceVoltageSource

Select the reference voltage source.

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

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

Initializes the eDMA peripheral.

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

Note: This function enables the minor loop map feature.

Parameters

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

void EDMA_Deinit(*EDMA_Type* *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

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

Push content of TCD structure into hardware TCD register.

Parameters

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

void EDMA__GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

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

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

Parameters

- config – A pointer to the eDMA configuration structure.

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

EDMA Channel initialization.

Parameters

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

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

Set channel memory attribute.

Parameters

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

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

Set channel sign extension.

Parameters

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

- `position` – A non-zero value specifying the sign extend bit position. If 0, sign extension is disabled.

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

Set channel swap size.

Parameters

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

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

Set channel access type.

Parameters

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

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

Set channel request source.

Parameters

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

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

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

Parameters

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

Returns

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

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

Set channel master ID replication.

Parameters

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

- enable – true is enable, false is disable.

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

Set channel security level.

Parameters

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

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

Sets all TCD registers to default values.

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

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Configures the eDMA transfer attribute.

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

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

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

Parameters

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

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

Configures the eDMA minor offset feature.

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

Parameters

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

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

Configures the eDMA channel preemption feature.

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

Parameters

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

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

Sets the channel link for the eDMA transfer.

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

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

Parameters

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

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

Sets the bandwidth for the eDMA transfer.

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

Parameters

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

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

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

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

Parameters

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

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

Enables an auto stop request for the eDMA transfer.

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

Parameters

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

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

Enables the interrupt source for the eDMA transfer.

Parameters

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

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

Disables the interrupt source for the eDMA transfer.

Parameters

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

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

Configures the eDMA channel TCD major offset feature.

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

Parameters

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

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

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

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

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

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

Note: This function enables the auto stop request feature.

Parameters

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

```
void EDMA_TcdReset(edma_tcd_t *tcd)
```

Sets all fields to default values for the TCD structure.

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

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

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

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

Configures the eDMA TCD transfer attribute.

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

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address

offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

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

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

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

Parameters

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

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

Configures the eDMA TCD minor offset feature.

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

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

Parameters

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

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

Sets the channel link for the eDMA TCD.

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

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

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

Parameters

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

- kEDMA_MinorLink
- kEDMA_MajorLink
- linkedChannel – The linked channel number.

static inline void EDMA_TcdSetBandWidth(*edma_tcd_t* *tcd, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

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

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

Parameters

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

void EDMA_TcdSetModulo(*edma_tcd_t* *tcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

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

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

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

Parameters

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

static inline void EDMA_TcdEnableAutoStopRequest(*edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

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

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

Parameters

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

void EDMA_TcdEnableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

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

Parameters

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

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

Disables the interrupt source for the eDMA TCD.

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

Parameters

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

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

Configures the eDMA TCD major offset feature.

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

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

Parameters

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

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

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

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

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Sets all fields to default values for the TCD structure.

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Configures the eDMA TCD transfer attribute.

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

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

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

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

Parameters

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

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

Configures the eDMA TCD minor offset feature.

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

Parameters

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

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

Sets the channel link for the eDMA TCD.

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

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

Parameters

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

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

Sets the bandwidth for the eDMA TCD.

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

Parameters

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

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

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

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

Parameters

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

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

Sets the auto stop request for the eDMA TCD.

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

Parameters

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

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

Enables the interrupt source for the eDMA TCD.

Parameters

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

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

Disables the interrupt source for the eDMA TCD.

Parameters

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

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

Configures the eDMA TCD major offset feature.

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

Parameters

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

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

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

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

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

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

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

static inline void EDMA_TriggerChannelStart(*EDMA_Type* *base, uint32_t channel)

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

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

uint32_t EDMA_GetRemainingMajorLoopCount(*EDMA_Type* *base, uint32_t channel)

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

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

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

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

Parameters

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

Returns

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

static inline uint32_t EDMA_GetErrorStatusFlags(*EDMA_Type* *base)

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

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

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

Gets the eDMA channel status flags.

Parameters

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

Returns

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

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

Clears the eDMA channel status flags.

Parameters

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

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

Creates the eDMA handle.

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

Parameters

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

Return values

- `kStatus_Success` –
- `kStatus_InvalidArgument` –

```
void EDMA__InstallTCDDMemory(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, ...)
```

(continues on next page)

(continued from previous page)

```
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_t)(struct _edma_handle *handle, void *userData, bool transferDone,
uint32_t tcDs)
```

Define callback function for eDMA.

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

Param handle

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

Param userData

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

Param transferDone

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

Param tcds

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

```
typedef struct _edma_handle edma_handle_t
    eDMA transfer handle structure

FSL_EDMA_DRIVER_EDMA4
    eDMA driver name

EDMA_ALLOCATE_TCD(name, number)
    Macro used for allocate edma TCD.

DMA_DCHPRI_INDEX(channel)
    Compute the offset unit from DCHPRI3.

struct _edma_channel_Preemption_config
    #include <fsl_edma.h> eDMA channel priority configuration
```

Public Members

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

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

uint8_t channelPriority
    Channel priority

struct _edma_minor_offset_config
    #include <fsl_edma.h> eDMA minor offset configuration
```

Public Members

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

bool enableDestMinorOffset

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

uint32_t minorOffset

Offset for a minor loop mapping.

struct _edma_channel_config

#include <fsl_edma.h> eDMA4 channel configuration

Public Members

edma_channel_preemption_config_t channelPreemptionConfig
channel preemption configuration

edma_channel_memory_attribute_t channelReadMemoryAttribute
channel memory read attribute configuration

edma_channel_memory_attribute_t channelWriteMemoryAttribute
channel memory write attribute configuration

edma_channel_swap_size_t channelSwapSize
channel swap size configuration

edma_channel_access_type_t channelAccessType
channel access type configuration

uint8_t channelDataSignExtensionBitPosition
channel data sign extension bit position configuration

uint32_t channelRequestSource
hardware service request source for the channel

bool enableMasterIDReplication
enable master ID replication

edma_channel_protection_level_t protectionLevel
protection level

struct _edma_transfer_config

#include <fsl_edma.h> edma4 channel transfer configuration

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

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

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

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

- 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.

2.8 eDMA core Driver

`enum _edma_tcd_type`

eDMA tcd flag type

Values:

enumerator `kEDMA_EDMA4Flag`

Data access for eDMA4 transfers.

enumerator `kEDMA_EDMA5Flag`

Instruction access for eDMA4 transfers.

`typedef struct _edma_core_mp edma_core_mp_t`

edma core channel struture definition

`typedef struct _edma_core_channel edma_core_channel_t`

edma core channel struture definition

`typedef enum _edma_tcd_type edma_tcd_type_t`

eDMA tcd flag type

`typedef struct _edma5_core_tcd edma5_core_tcd_t`

edma5 core TCD struture definition

`typedef struct _edma4_core_tcd edma4_core_tcd_t`

edma4 core TCD struture definition

```
typedef struct edma_core_tcd edma_core_tcd_t
    edma core TCD struture definition
typedef edma_core_channel_t EDMA_ChannelType
    EDMA typedef.
typedef edma_core_tcd_t EDMA_TCDType
typedef void EDMA_Type

DMA_CORE_MP_CSR_EDBG_MASK
DMA_CORE_MP_CSR_ERCA_MASK
DMA_CORE_MP_CSR_HAE_MASK
DMA_CORE_MP_CSR_HALT_MASK
DMA_CORE_MP_CSR_GCLC_MASK
DMA_CORE_MP_CSR_GMRC_MASK
DMA_CORE_MP_CSR_EDBG(x)
DMA_CORE_MP_CSR_ERCA(x)
DMA_CORE_MP_CSR_HAE(x)
DMA_CORE_MP_CSR_HALT(x)
DMA_CORE_MP_CSR_GCLC(x)
DMA_CORE_MP_CSR_GMRC(x)
DMA_CSR_INTMAJOR_MASK
DMA_CSR_INTHALF_MASK
DMA_CSR_DREQ_MASK
DMA_CSR_ESG_MASK
DMA_CSR_BWC_MASK
DMA_CSR_BWC(x)
DMA_CSR_START_MASK
DMA_CITER_ELINKNO_CITER_MASK
DMA_BITER_ELINKNO_BITER_MASK
DMA_CITER_ELINKNO_CITER_SHIFT
DMA_CITER_ELINKYES_CITER_MASK
DMA_CITER_ELINKYES_CITER_SHIFT
DMA_ATTR_SMOD_MASK
DMA_ATTR_DMOD_MASK
DMA_CITER_ELINKNO_ELINK_MASK
DMA_CSR_MAJORELINK_MASK
```

DMA_BITER_ELINKYES_ELINK_MASK
DMA_CITER_ELINKYES_ELINK_MASK
DMA_CSR_MAJORLINKCH_MASK
DMA_BITER_ELINKYES_LINKCH_MASK
DMA_CITER_ELINKYES_LINKCH_MASK
DMA_NBYTES_MLOFFYES_MLOFF_MASK
DMA_NBYTES_MLOFFYES_DMLOE_MASK
DMA_NBYTES_MLOFFYES_SMLOE_MASK
DMA_NBYTES_MLOFFNO_NBYTES_MASK
DMA_ATTR_DMOD(x)
DMA_ATTR_SMOD(x)
DMA_BITER_ELINKYES_LINKCH(x)
DMA_CITER_ELINKYES_LINKCH(x)
DMA_NBYTES_MLOFFYES_MLOFF(x)
DMA_NBYTES_MLOFFYES_DMLOE(x)
DMA_NBYTES_MLOFFYES_SMLOE(x)
DMA_NBYTES_MLOFFNO_NBYTES(x)
DMA_NBYTES_MLOFFYES_NBYTES(x)
DMA_ATTR_DSIZE(x)
DMA_ATTR_SSIZE(x)
DMA_CSR_DREQ(x)
DMA_CSR_MAJORLINKCH(x)
DMA_CH_MATTR_WCACHE(x)
DMA_CH_MATTR_RCACHE(x)
DMA_CH_CSR_SIGNEXT_MASK
DMA_CH_CSR_SIGNEXT_SHIFT
DMA_CH_CSR_SWAP_MASK
DMA_CH_CSR_SWAP_SHIFT
DMA_CH_SBR_INSTR_MASK
DMA_CH_SBR_INSTR_SHIFT
DMA_CH_SBR_EMI_MASK
DMA_CH_SBR_EMI_SHIFT
DMA_CH_MUX_SOURCE(x)

DMA_ERR_DBE_FLAG

DMA error flag.

DMA_ERR_SBE_FLAG

DMA_ERR_SGE_FLAG

DMA_ERR_NCE_FLAG

DMA_ERR_DOE_FLAG

DMA_ERR_DAE_FLAG

DMA_ERR_SOE_FLAG

DMA_ERR_SAE_FLAG

DMA_ERR_ERRCHAN_FLAG

DMA_ERR_ECX_FLAG

DMA_ERR_FLAG

DMA_CLEAR_DONE_STATUS(base, channel)

get/clear DONE bit

DMA_GET_DONE_STATUS(base, channel)

DMA_ENABLE_ERROR_INT(base, channel)

enable/disable error interrupt

DMA_DISABLE_ERROR_INT(base, channel)

DMA_CLEAR_ERROR_STATUS(base, channel)

get/clear error status

DMA_GET_ERROR_STATUS(base, channel)

DMA_CLEAR_INT_STATUS(base, channel)

get/clear INT status

DMA_GET_INT_STATUS(base, channel)

DMA_ENABLE_MAJOR_INT(base, channel)

enable/disable MAJOR/HALF INT

DMA_ENABLE_HALF_INT(base, channel)

DMA_DISABLE_MAJOR_INT(base, channel)

DMA_DISABLE_HALF_INT(base, channel)

EDMA_TCD_ALIGN_SIZE

EDMA tcd align size.

EDMA_CORE_BASE(base)

EDMA base address convert macro.

EDMA_MP_BASE(base)

EDMA_CHANNEL_BASE(base, channel)

EDMA_TCD_BASE(base, channel)

EDMA_TCD_TYPE(x)

EDMA TCD type macro.

EDMA_TCD_SADDR(tcd, flag)

EDMA TCD address convert macro.

EDMA_TCD_SOFF(tcd, flag)

EDMA_TCD_ATTR(tcd, flag)

EDMA_TCD_NBYTES(tcd, flag)

EDMA_TCD_SLAST(tcd, flag)

EDMA_TCD_DADDR(tcd, flag)

EDMA_TCD_DOFF(tcd, flag)

EDMA_TCD_CITER(tcd, flag)

EDMA_TCD_DLAST_SGA(tcd, flag)

EDMA_TCD_CSR(tcd, flag)

EDMA_TCD_BITER(tcd, flag)

struct _edma_core_mp

#include <fsl_edma_core.h> edma core channel struture definition

Public Members

__IO uint32_t MP_CSR

Channel Control and Status, array offset: 0x10000, array step: 0x10000

__IO uint32_t MP_ES

Channel Error Status, array offset: 0x10004, array step: 0x10000

struct _edma_core_channel

#include <fsl_edma_core.h> edma core channel struture definition

Public Members

__IO uint32_t CH_CSR

Channel Control and Status, array offset: 0x10000, array step: 0x10000

__IO uint32_t CH_ES

Channel Error Status, array offset: 0x10004, array step: 0x10000

__IO uint32_t CH_INT

Channel Interrupt Status, array offset: 0x10008, array step: 0x10000

__IO uint32_t CH_SBR

Channel System Bus, array offset: 0x1000C, array step: 0x10000

__IO uint32_t CH_PRI

Channel Priority, array offset: 0x10010, array step: 0x10000

struct _edma5_core_tcd

#include <fsl_edma_core.h> edma5 core TCD struture definition

Public Members

__IO uint32_t SADDR
SADDR register, used to save source address

__IO uint32_t SADDR_HIGH
SADDR HIGH register, used to save source address

__IO uint16_t SOFF
SOFF register, save offset bytes every transfer

__IO uint16_t ATTR
ATTR register, source/destination transfer size and modulo

__IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

__IO uint32_t SLAST
SLAST register

__IO uint32_t SLAST_SDA_HIGH
SLAST SDA HIGH register

__IO uint32_t DADDR
DADDR register, used for destination address

__IO uint32_t DADDR_HIGH
DADDR HIGH register, used for destination address

__IO uint32_t DLAST_SGA
DLASTSGA register, next tcd address used in scatter-gather mode

__IO uint32_t DLAST_SGA_HIGH
DLASTSGA HIGH register, next tcd address used in scatter-gather mode

__IO uint16_t DOFF
DOFF register, used for destination offset

__IO uint16_t CITER
CITER register, current minor loop numbers, for unfinished minor loop.

__IO uint16_t CSR
CSR register, for TCD control status

__IO uint16_t BITER
BITER register, begin minor loop count.

uint8_t RESERVED[16]
Aligned 64 bytes

struct _edma4_core_tcd
#include <fsl_edma_core.h> edma4 core TCD struture definition

Public Members

__IO uint32_t SADDR
SADDR register, used to save source address

__IO uint16_t SOFF
SOFF register, save offset bytes every transfer


```

__IO uint16_t ATTR
    ATTR register, source/destination transfer size and modulo
__IO uint32_t NBYTES
    Nbytes register, minor loop length in bytes
__IO uint32_t SLAST
    SLAST register
__IO uint32_t DADDR
    DADDR register, used for destination address
__IO uint16_t DOFF
    DOFF register, used for destination offset
__IO uint16_t CITER
    CITER register, current minor loop numbers, for unfinished minor loop.
__IO uint32_t DLAST_SGA
    DLASTSGA register, next tcd address used in scatter-gather mode
__IO uint16_t CSR
    CSR register, for TCD control status
__IO uint16_t BITER
    BITER register, begin minor loop count.
struct _edma_core_tcd
    #include <fsl_edma_core.h> edma core TCD struture definition
union MP_REGS

```

Public Members

```

struct _edma_core_mp MDMA5_REG
struct MDMA5_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
```

2.9 eDMA soc Driver

```
void EDMA_SocRequestEnable(DMA_Type *base, uint32_t request, bool value)
```

Enable or disable EDMA request for SoC.

This function enables or disables a specific EDMA request by setting or clearing the corresponding bit in the AHBSC security general purpose register.

Parameters

- *base* – DMA peripheral base address (DMA0 or other DMA instance).
- *request* – The EDMA request number to enable or disable.
- *value* – true to enable the request, false to disable it.

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)

2.10 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.

2.11 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.

2.12 eSPI: Enhanced Serial Peripheral Interface

enum _espi_port_type
eSPI PnCFG.TYPE register values.

Values:

enumerator kESPI_PortType_Unconfigured
0000b: Unconfigured (reset condition).

enumerator kESPI_PortType_ACPIEndpoint
0001b: ACPI style Endpoint.

enumerator kESPI_PortType_ACPIIndexData
0010b: ACPI style Index/Data.

enumerator kESPI_PortType_BusMasterMemSingle
0100b: Bus Master Mem Single.

enumerator kESPI_PortType_BusMasterFlashSingle
0101b: Bus Master Flash Single.

enumerator kESPI_PortType_MailboxShared
1000b: Mailbox Shared.

enumerator kESPI_PortType_MailboxSingle
1001b: Mailbox Single.

enumerator kESPI_PortType_MailboxSplit
1010b: Mailbox Split.

enumerator kESPI_PortType_MailboxOOBSplit
1011b: Mailbox OOB Split.

enumerator kESPI_PortType_MailboxOEM
1100b: Mailbox OEM.

enum _espi_saf_rx_completion_type
eSPI SAF read completion types.

Values:

enumerator kESPI_SAFReadMiddle
Middle completion of split sequence.

enumerator kESPI_SAFReadFirst
First completion of split sequence.

enumerator kESPI_SAFReadLast
Last completion of split sequence.

enumerator kESPI_SAFReadOnly

Only completion for single transaction.

enum __espi_flash_transaction_type

eSPI flash transaction types.

Values:

enumerator kESPI_FlashTransNone

No transaction.

enumerator kESPI_FlashTransRead

Read transaction.

enumerator kESPI_FlashTransWrite

Write transaction.

enumerator kESPI_FlashTransErase

Erase transaction.

enum __espi_enable_mode

eSPI/LPC enable mode encoding.

Values:

enumerator kESPI_Disable

enumerator kESPI_EnableESPI

enumerator kESPI_EnableLPC

enum __espi_read_req_size

Values:

enumerator kESPI_ReadReq64B

enumerator kESPI_ReadReq128B

enumerator kESPI_ReadReq256B

enumerator kESPI_ReadReq512B

enumerator kESPI_ReadReq1024B

enumerator kESPI_ReadReq2048B

enumerator kESPI_ReadReq4096B

enum __espi_mem_max_payload

Values:

enumerator kESPI_MemMax64B

enumerator kESPI_MemMax128B

enumerator kESPI_MemMax256B

enum __espi_flash_max_payload

Values:

enumerator kESPI_FlashMax64B

enumerator kESPI_FlashMax128B

enumerator kESPI_FlashMax256B

enumerator kESPI_FlashMax512B

enum __espi_oob_max_payload

Values:

enumerator kESPI_OOBMax64B

enumerator kESPI_OOBMax128B

enumerator kESPI_OOBMax256B

enum __espi_spi_mode

eSPI SPI capability mode.

Defines the SPI wire mode capabilities supported by the eSPI controller. These values are programmed into ESPICAP.SPICAP field to indicate which SPI modes the target supports to the host.

Mode encoding.

- Single: Standard SPI using single data line (MOSI/MISO).
- Dual: Dual SPI using 2 data lines (IO[1:0]).
- Quad: Quad SPI using 4 data lines (IO[3:0]).

Values:

enumerator kESPI_SPIMode_SingleOnly

Single SPI only.

enumerator kESPI_SPIMode_SingleAndDual

Single + Dual SPI.

enumerator kESPI_SPIMode_SingleAndQuad

Single + Quad SPI.

enumerator kESPI_SPIMode_All

All modes: Single, Dual and Quad SPI

enum __espi_wrstat_values

eSPI WRSTAT field values (PnSTAT[WRSTAT] or STAT.WRSTAT).

These symbolic names document the 2-bit WRSTAT encoding used by the eSPI controller to indicate host flash request types. The numeric encoding comes from the eSPI specification; use these symbols in the driver instead of magic numbers for clarity.

Values:

enumerator kESPI_WRSTAT_NoRequest

00b: No flash request / idle.

enumerator kESPI_WRSTAT_ReadRequest

01b: Host read request.

enumerator kESPI_WRSTAT_WriteRequest

10b: Host write request.

enumerator kESPI_WRSTAT_EraseRequest

11b: Host erase request.

enum __espi_interrupt_enable

eSPI interrupt enable flags.

Values:

enumerator kESPI_PortInterruptEnable

Port interrupt enable.

enumerator kESPI_Port80InterruptEnable

Port 80 interrupt enable.

enumerator kESPI_BusResetInterruptEnable

Bus reset interrupt enable.

enumerator kESPI_IrqUpdateInterruptEnable

IRQ update interrupt enable.

enumerator kESPI_WireChangeInterruptEnable

Wire change interrupt enable.

enumerator kESPI_HostStallInterruptEnable

Host stall interrupt enable.

enumerator kESPI_CrcErrorInterruptEnable

CRC error interrupt enable.

enumerator kESPI_GpioInterruptEnable

GPIO interrupt enable.

enumerator kESPI-CsInterruptEnable

CS interrupt enable.

enumerator kESPI_AllInterruptEnable

All interrupts enable.

enum _espi_status_flags

eSPI status flags.

Values:

enumerator kESPI_PortInterruptFlag

Port interrupt flag.

enumerator kESPI_Port80InterruptFlag

Port 80 interrupt flag.

enumerator kESPI_BusResetFlag

Bus reset flag.

enumerator kESPI_IrqUpdateFlag

IRQ update flag.

enumerator kESPI_WireChangeFlag

Wire change flag.

enumerator kESPI_HostStallFlag

Host stall flag.

enumerator kESPI_CrcErrorFlag

CRC error flag.

enumerator kESPI_GpioFlag

GPIO flag.

enumerator kESPI_BusyFlag

Bus busy flag.

enumerator kESPI_InResetFlag

In reset flag.

enumerator kESPI_CompletionPendingFlag

Completion pending flag.

enumerator kESPI_MasterPendingFlag

Master pending flag.

enumerator kESPI_AlertPendingFlag

Alert pending flag.

enumerator kESPI_BusStatusFlags

enum __espi_port_interrupt_flags

eSPI port interrupt flags.

Values:

enumerator kESPI_PortErrorInterrupt

Error interrupt.

enumerator kESPI_PortReadInterrupt

Read interrupt.

enumerator kESPI_PortWriteInterrupt

Write interrupt.

enumerator kESPI_PortSpec0Interrupt

Special 0 interrupt.

enumerator kESPI_PortSpec1Interrupt

Special 1 interrupt.

enumerator kESPI_PortSpec2Interrupt

Special 2 interrupt.

enumerator kESPI_PortSpec3Interrupt

Special 3 interrupt.

enumerator kESPI_PortAllInterrupts

enum __espi_saf_erase_size

eSPI SAF erase size enumeration

Values:

enumerator kESPI_SAFEraser2KB

2 kB minimum erase sector.

enumerator kESPI_SAFEraser4KB

4 kB minimum erase sector.

enumerator kESPI_SAFEraser8KB

8 kB minimum erase sector.

enumerator kESPI_SAFEraser16KB

16 kB minimum erase sector.

enum __espi_max_speed

eSPI maximum SPI speed enumeration

Values:

enumerator kESPI_MaxSpeed20MHz
 <= 20 MHz.

enumerator kESPI_MaxSpeed25MHz
 <= 25 MHz (24 MHz).

enumerator kESPI_MaxSpeed33MHz
 <= 33 MHz (30 MHz).

enumerator kESPI_MaxSpeed50MHz
 <= 50 MHz (48 MHz).

enumerator kESPI_MaxSpeed66MHz
 <= 66 MHz (60 MHz).

enum _espi_ram_size
 eSPI RAM length.

Values:

enumerator kESPI_RamSize4B

enumerator kESPI_RamSize8B

enumerator kESPI_RamSize16B

enumerator kESPI_RamSize32B

enumerator kESPI_RamSize64B

enumerator kESPI_RamSize128B

enumerator kESPI_RamSize256B

enumerator kESPI_RamSize512B

enum _espi_addr_base
 eSPI address base selection.

Determines how the port address is calculated:

- Direct: Address = PnADDR[OFF]
- Base0: Address = (MAPBASE[BASE0] « 16) | PnADDR[OFF]
- Base1: Address = (MAPBASE[BASE1] « 16) | PnADDR[OFF]

Values:

enumerator kESPI_AddrBaseDirect

 Direct addressing: 0x0000-0xFFFF (I/O or low memory)

enumerator kESPI_AddrBase0

 Use MAPBASE[BASE0] as upper 16 bits

enumerator kESPI_AddrBase1

 Use MAPBASE[BASE1] as upper 16 bits

enum _espi_sstcl
 eSPI PnIRULESTAT.SSTCL (Status Set and Clear) values.

These values are written to PnIRULESTAT[19:16] to control port status transitions. The meaning depends on the port type configured in PnCFG[TYPE].

For Endpoint/Index-and-Data types (TYPE=0x1, 0x2):

- Individual bits control specific status flags

- Bit 0: Set RDPEND
- Bit 1: Clear WRRDY
- Bit 2: Clear RDPEND
- Bit 3: Clear CMD (Endpoint only)

For Mailbox types (TYPE=0x8, 0x9, 0xA, 0xB):

- Bits [1:0] control RDSTAT (Host Read / MCU Write direction)
- Bits [3:2] control WRSTAT (Host Write / MCU Read direction)

For Bus Master types (TYPE=0x4, 0x5):

- Value 0x1: Start memory/flash request

For SAF (Slave-Attached Flash):

- Value 0x1: Send completion response

Values:

enumerator kESPI_SSTCL_ACPISetRdPend

Bit 0=1: Set RDPEND (read data pending).

enumerator kESPI_SSTCL_ACPIClrWrRdy

Bit 1=1: Clear WRRDY (write ready).

enumerator kESPI_SSTCL_ACPIClrRdPend

Bit 2=1: Clear RDPEND (read data pending).

enumerator kESPI_SSTCL_ACPIClrCmd

Bit 3=1: Clear CMD flag (Endpoint only).

enumerator kESPI_SSTCL_MailboxRdStarted

01b: MCU started writing data.

enumerator kESPI_SSTCL_MailboxRdCompleted

10b: MCU completed writing data.

enumerator kESPI_SSTCL_MailboxRdEmpty

11b: Set read status to Empty.

enumerator kESPI_SSTCL_MailboxWrStarted

01b « 2: MCU started reading (partial).

enumerator kESPI_SSTCL_MailboxWrEmpty

11b « 2: Set write status to Empty.

enumerator kESPI_SSTCL_MailboxBothEmpty

Clear both RDSTAT and WRSTAT to Empty.

enumerator kESPI_SSTCL_OOBSendStart

Start OOB message send.

enumerator kESPI_SSTCL_OOBSendComplete

Complete OOB send, trigger host GET.

enumerator kESPI_SSTCL_OOBSendEmpty

Clear OOB send status.

enumerator kESPI_SSTCL_OOBRecvEmpty

Clear OOB receive status.

enumerator kESPI_SSTCL_MAFStart
Start memory/flash request (MAF).

enumerator kESPI_SSTCL_SAFCompletion
Send SAF completion response.

enumerator kESPI_SSTCL_SAFReqAccepted
Flash request accepted.

enum __espi_omflen_trans

eSPI OMFLEN[TRANS] field values.

Encodes the 2-bit TRANS field of PnOMFLEN for different port types. The same numeric values have context-specific meanings across OOB, bus mastering (memory/MAF), and SAF completion.

Reference: MCXA577 eSPI RM – Port OOB, Mastering, and Flash Length (PnOMFLEN).

Values:

enumerator kESPI_OMFLEN_OOB_ToHost
00b: OOB message to host.

enumerator kESPI_OMFLEN_MasterToHost32
00b: To host, 32-bit address read.

enumerator kESPI_OMFLEN_MasterToHost64
01b: To host, 64-bit address read.

enumerator kESPI_OMFLEN_MasterFromHost32
10b: From host, 32-bit address write.

enumerator kESPI_OMFLEN_MasterFromHost64
11b: From host, 64-bit address write.

enumerator kESPI_OMFLEN_MAFReadFlash
01b: Read flash (location in RAM).

enumerator kESPI_OMFLEN_MAFWriteFlash
10b: Write/program flash (location in RAM).

enumerator kESPI_OMFLEN_MAFEraseFlash
11b: Erase flash (sector info in RAM).

enumerator kESPI_OMFLEN_SAFCompletionFail
00b: Completion failure (LEN=0).

enumerator kESPI_OMFLEN_SAFCompletionWithData
01b: Completion with data (read).

enumerator kESPI_OMFLEN_SAFCompletionNoData
10b: Completion with no data (write/erase).

enum __espi_port_error

eSPI port error codes (context-dependent on port type).

Values:

enumerator kESPI_PortError_None
No error.

enumerator kESPI_PortError_EndpointWriteOverrun
ERR0: Host wrote when WRDY=1.

enumerator kESPI_PortError_EndpointReadEmpty

ERR1: Host read when RPEND=0.

enumerator kESPI_PortError_EndpointInvalidSize

ERR2: Transfer size > 1 byte.

enumerator kESPI_PortError_MailboxInvalidAccess

ERR0: Invalid host read/write access.

enumerator kESPI_PortError_MailboxOverrunUnderrun

ERR1: Write overrun or read underrun.

enumerator kESPI_PortError_MailboxSizeOverflow

ERR2: Request size exceeds mailbox boundary.

enumerator kESPI_PortError_MailboxRAMBusError

ERR3: AHB/RAM access error.

enumerator kESPI_PortError_MasterFromHostFailed

ERR0: From-host transfer failed.

enumerator kESPI_PortError_MasterOverrunUnderrun

ERR1: Transfer timing error.

enumerator kESPI_PortError_MasterEraseFailed

ERR2: Flash erase failed.

enumerator kESPI_PortError_MasterBusError

ERR3: Bus master access error.

enum _espi_vw_rd_flags

eSPI Virtual Wire receive flags.

Symbolic names for bits reported by WIRERO. These indicate host-driven VWIRE signal states latched in the receive register.

Values:

enumerator kESPI_VWireRd_SlpS3N

Sleep State S3.

enumerator kESPI_VWireRd_SlpS4N

Sleep State S4.

enumerator kESPI_VWireRd_SlpS5N

Sleep State S5.

enumerator kESPI_VWireRd_SusStat

Suspend Status.

enumerator kESPI_VWireRd_PltRst

Platform Reset Request.

enumerator kESPI_VWireRd_OobRstWarn

OOB Reset Warning.

enumerator kESPI_VWireRd_HostRstWarn

Host Reset Warning.

enumerator kESPI_VWireRd_SusWarn

Suspend Warning.

enumerator kESPI_VWireRd_SusPwrAck

Suspend Power Well Acknowledge.

enumerator kESPI_VWireRd_SlpAN
Sleep AN.

enumerator kESPI_VWireRd_SlpLAN
Wired LAN Sleep.

enumerator kESPI_VWireRd_SlpWLAN
Wireless LAN Sleep.

enumerator kESPI_VWireRd_P2E
PCIe to EC group.

enumerator kESPI_VWireRd_HostC10N
Host entering C10 state.

enum _espi_vw_wr_flags

eSPI Virtual Wire write/ack flags (WIREWO)

Symbolic names for bits written to WIREWO to drive or acknowledge VWIRE signals to the host.

Values:

enumerator kESPI_VWireWr_DswPwrokRst
DSW / PWR OK / Reset (Wr).

enumerator kESPI_VWireWr_BootErrn
Boot error (active low, Wr).

enumerator kESPI_VWireWr_BootDone
Boot done (Wr).

enumerator kESPI_VWireWr_E2P
E2P field (Wr).

enumerator kESPI_VWireWr_SusAckN
Suspend acknowledge (active low, Wr).

enumerator kESPI_VWireWr_HostRstAck
Host reset acknowledge (Wr).

enumerator kESPI_VWireWr_Rcinn
RCINN group (Wr).

enumerator kESPI_VWireWr_Smin
SMIN group (Wr).

enumerator kESPI_VWireWr_Scin
SCIN group (Wr).

enumerator kESPI_VWireWr_Pmen
Power management enable (Wr).

enumerator kESPI_VWireWr_WakenScin
Wake enable / sensor input (Wr).

enumerator kESPI_VWireWr_OobRstAck
OOB reset acknowledge (Wr).

typedef enum _espi_port_type espi_port_type_t
eSPI PnCFG.TYPE register values.

typedef enum _espi_saf_rx_completion_type espi_saf_rx_completion_type_t
eSPI SAF read completion types.

typedef enum *_espi_flash_transaction_type* espi_flash_trans_type_t
eSPI flash transaction types.

typedef enum *_espi_enable_mode* espi_enable_mode_t
eSPI/LPC enable mode encoding.

typedef enum *_espi_read_req_size* espi_read_req_size_t

typedef enum *_espi_mem_max_payload* espi_mem_max_payload_t

typedef enum *_espi_flash_max_payload* espi_flash_max_payload_t

typedef enum *_espi_oob_max_payload* espi_oob_max_payload_t

typedef enum *_espi_spi_mode* espi_spi_mode_t
eSPI SPI capability mode.

Defines the SPI wire mode capabilities supported by the eSPI controller. These values are programmed into ESPICAP.SPICAP field to indicate which SPI modes the target supports to the host.

Mode encoding.

- Single: Standard SPI using single data line (MOSI/MISO).
- Dual: Dual SPI using 2 data lines (IO[1:0]).
- Quad: Quad SPI using 4 data lines (IO[3:0]).

typedef enum *_espi_wrstat_values* espi_wrstat_values_t
eSPI WRSTAT field values (PnSTAT[WRSTAT] or STAT.WRSTAT).

These symbolic names document the 2-bit WRSTAT encoding used by the eSPI controller to indicate host flash request types. The numeric encoding comes from the eSPI specification; use these symbols in the driver instead of magic numbers for clarity.

typedef enum *_espi_saf_erase_size* espi_saf_erase_size_t
eSPI SAF erase size enumeration

typedef enum *_espi_max_speed* espi_max_speed_t
eSPI maximum SPI speed enumeration

typedef enum *_espi_ram_size* espi_ram_size_t
eSPI RAM length.

typedef enum *_espi_addr_base* espi_addr_base_t
eSPI address base selection.

Determines how the port address is calculated:

- Direct: Address = PnADDR[OFF]
- Base0: Address = (MAPBASE[BASE0] « 16) | PnADDR[OFF]
- Base1: Address = (MAPBASE[BASE1] « 16) | PnADDR[OFF]

typedef enum *_espi_sstcl* espi_sstcl_t
eSPI PnIRULESTAT.SSTCL (Status Set and Clear) values.

These values are written to PnIRULESTAT[19:16] to control port status transitions. The meaning depends on the port type configured in PnCFG[TYPE].

For Endpoint/Index-and-Data types (TYPE=0x1, 0x2):

- Individual bits control specific status flags
- Bit 0: Set RDPEND
- Bit 1: Clear WRRDY

- Bit 2: Clear RDPEND
- Bit 3: Clear CMD (Endpoint only)

For Mailbox types (TYPE=0x8, 0x9, 0xA, 0xB):

- Bits [1:0] control RDSTAT (Host Read / MCU Write direction)
- Bits [3:2] control WRSTAT (Host Write / MCU Read direction)

For Bus Master types (TYPE=0x4, 0x5):

- Value 0x1: Start memory/flash request

For SAF (Slave-Attached Flash):

- Value 0x1: Send completion response

`typedef enum _espi_omflen_trans espi_omflen_trans_t`

eSPI OMFLEN[TRANS] field values.

Encodes the 2-bit TRANS field of PnOMFLEN for different port types. The same numeric values have context-specific meanings across OOB, bus mastering (memory/MAF), and SAF completion.

Reference: MCXA577 eSPI RM – Port OOB, Mastering, and Flash Length (PnOMFLEN).

`typedef enum _espi_port_error espi_port_error_t`

eSPI port error codes (context-dependent on port type).

`typedef struct _espi_port_config espi_port_config_t`

eSPI per-port configuration structure (PnCFG/PnADDR/PnRAMUSE).

This describes one hardware port (Pn). ESPI_Init will program CFG, RAMUSE and ADDR for each entry if provided in `espi_config_t::portConfig`.

`typedef struct _espi_config espi_config_t`

eSPI configuration structure.

`typedef struct _espi_port80_status espi_p80_status_t`

Port 80 status structure.

`typedef enum _espi_vw_rd_flags espi_vw_rd_flags_t`

eSPI Virtual Wire receive flags.

Symbolic names for bits reported by WIRERO. These indicate host-driven VWIRE signal states latched in the receive register.

`typedef enum _espi_vw_wr_flags espi_vw_wr_flags_t`

eSPI Virtual Wire write/ack flags (WIREWO)

Symbolic names for bits written to WIREWO to drive or acknowledge VWIRE signals to the host.

`typedef struct _espi_flash_request espi_flash_request_t`

eSPI flash request structure.

`typedef struct _espi_handle espi_handle_t`

`typedef void (*espi_common_callback_t)(ESPI_Type *base, uint32_t status, void *userData)`

eSPI common callback function pointer.

`typedef void (*espi_port_callback_t)(ESPI_Type *base, espi_handle_t *handle, uint32_t port, uint32_t status, void *userData)`

eSPI port callback function pointer.


```
typedef void (*espi_flash_ops_t)(ESPI_Type *base, espi_handle_t *handle, espi_flash_request_t *req, void *userData)
```

eSPI flash callback function pointer.

```
typedef struct _espi_callback_config espi_callback_config_t
```

eSPI callback configuration structure.

```
FSL_ESPI_DRIVER_VERSION
```

```
ESPI_INVALID_PORT
```

```
void ESPI_Init(ESPI_Type *base, const espi_config_t *config)
```

Initializes the eSPI peripheral.

This function enables the eSPI clock, configures the eSPI peripheral according to the configuration structure.

Parameters

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

```
void ESPI_Deinit(ESPI_Type *base)
```

Deinitializes the eSPI peripheral.

This function disables the eSPI peripheral and gates the eSPI clock.

Parameters

- base – eSPI peripheral base address.

```
void ESPI_GetDefaultConfig(espi_config_t *config)
```

Gets the default configuration structure.

This function initializes the eSPI configuration structure to default values.

Parameters

- config – Pointer to the configuration structure.

```
static inline uint32_t ESPI_GetStatusFlags(ESPI_Type *base)
```

Gets the eSPI status flags.

Parameters

- base – eSPI peripheral base address.

Returns

Status flags mask. AND with `_espi_status_flags` to test specific flags.

```
static inline void ESPI_ClearStatusFlags(ESPI_Type *base, uint32_t mask)
```

Clears the eSPI status flags.

Parameters

- base – eSPI peripheral base address.
- mask – Status flag mask, see `_espi_status_flags`.

```
static inline uint8_t *ESPI_GetPortRamBuffer(ESPI_Type *base, uint32_t port)
```

Gets a pointer to the RAM memory buffer.

Returns a pointer to the start of the mailbox RAM for a port.

Parameters

- base – eSPI peripheral base address.
- port – Hardware port index.

Returns

Pointer to the start of the mailbox/port RAM.

static inline uint32_t ESPI_GetFlashMaxPayload(ESPI_Type *base)

Gets the maximum flash payload size.

Calculates the configured maximum flash payload in bytes.

Parameters

- base – eSPI peripheral base address.

Returns

Maximum flash payload size in bytes.

static inline void ESPI_EnableInterrupts(ESPI_Type *base, uint32_t mask)

Enables the eSPI interrupts.

Parameters

- base – eSPI peripheral base address.
- mask – Interrupt source, see `_espi_interrupt_enable`.

static inline void ESPI_DisableInterrupts(ESPI_Type *base, uint32_t mask)

Disables the eSPI interrupts.

Parameters

- base – eSPI peripheral base address.
- mask – Interrupt source, see `_espi_interrupt_enable`.

static inline uint32_t ESPI_GetEnabledInterrupts(ESPI_Type *base)

Gets the enabled eSPI interrupts.

Parameters

- base – eSPI peripheral base address.

Returns

Enabled interrupts, see `_espi_interrupt_enable`.

static inline void ESPI_PushIrq(ESPI_Type *base, uint8_t irqNum)

Sets the IRQ push register.

This register allows the application to push an IRQ assert through eSPI.

Parameters

- base – eSPI peripheral base address.
- irqNum – IRQ to push across to host.

static inline bool ESPI_IsIrqPushDone(ESPI_Type *base)

Check if IRQ Push is Done.

Parameters

- base – eSPI peripheral base address.

static inline *espi_port_type_t* ESPI_GetPortType(ESPI_Type *base, uint32_t port)

Gets the configured type of an eSPI port.

This function reads the PnCFG register and extracts the TYPE field, which indicates how the port interacts with the host (endpoint, mailbox, bus master, etc.).

Parameters

- base – eSPI peripheral base address.

- port – Port index (0 to ESPI_PORT_COUNT-1).

Returns

Port type value, see `espi_port_type_t`.

`static inline void ESPI_EnablePortInterrupts(ESPI_Type *base, uint32_t port, uint32_t mask)`
Enables the eSPI port interrupts.

Parameters

- base – eSPI peripheral base address.
- port – eSPI port.
- mask – Interrupt mask, see `_espi_port_interrupt_flags`.

`static inline void ESPI_DisablePortInterrupts(ESPI_Type *base, uint32_t port, uint32_t mask)`
Disables the eSPI port interrupts.

Parameters

- base – eSPI peripheral base address.
- port – eSPI port.
- mask – Interrupt mask, see `_espi_port_interrupt_flags`.

`static inline uint32_t ESPI_GetPortStatus(ESPI_Type *base, uint32_t port)`
Gets the eSPI port status.

Parameters

- base – eSPI peripheral base address.
- port – eSPI port index.

Returns

Port status flags.

`static inline void ESPI_WritePortData(ESPI_Type *base, uint32_t port, uint32_t data)`
Writes the eSPI port DATAOUT register.

This function writes a response byte to the DATAOUT register for endpoint or index/data port types.

Parameters

- base – eSPI peripheral base address.
- port – eSPI port index.
- data – Value to write to DATAOUT.

`static inline void ESPI_ClearPortStatus(ESPI_Type *base, uint32_t port, uint32_t mask)`
Clears the eSPI port status flags.

Parameters

- base – eSPI peripheral base address.
- port – eSPI port index.
- mask – Port interrupt status flag mask, see `_espi_port_interrupt_flags`.

`static inline uint32_t ESPI_GetPortMailboxSize(ESPI_Type *base, uint32_t port)`
Gets the mailbox maximum size (bytes) for a port.

Parameters

- base – eSPI peripheral base address.
- port – Hardware port index.

Returns

Mailbox size in bytes.

status_t ESPI_GetPort80Status(ESPI_Type *base, *espi_p80_status_t* *status)

Gets Port 80 status and POST codes.

This function reads the Port 80 status register and extracts the current POST code, previous POST code, and counter value. Port 80 is used by BIOS/UEFI to write Power-On Self-Test (POST) codes during boot for debugging purposes.

Parameters

- base – eSPI peripheral base address.
- status – Pointer to structure to receive Port 80 status.

Return values

- kStatus_Success – Successfully read Port 80 status.
- kStatus_Fail – Port 80 is not enabled.

static inline void ESPI_ResetPort80Counter(ESPI_Type *base)

Resets Port 80 counter.

This function resets the Port 80 POST code counter back to 0.

Parameters

- base – eSPI peripheral base address.

void ESPI_GetEndpointData(ESPI_Type *base, uint32_t port, uint32_t *idx, uint32_t *data)

Gets endpoint data fields from a port.

This function reads PnDATAIN and extracts the IDX field and DATA_LEN field.

Parameters

- base – eSPI peripheral base address.
- port – eSPI port index.
- idx – Pointer to store the extracted IDX field.
- data – Pointer to store the extracted DATA_LEN field.

void ESPI_GetPortErrorStatus(ESPI_Type *base, uint32_t port, uint32_t pstat, *espi_port_error_t* *error)

Checks port error status.

Reads and interprets the error bits (ERR0-ERR3) from PnSTAT register. Returns a specific error code based on the port type.

Error bit interpretation depends on port type.

Note: Error status bits are read-only and automatically cleared by hardware when the error condition is resolved.

Note: Only returns the first error found (priority: ERR0 > ERR1 > ERR2 > ERR3).

Parameters

- base – eSPI peripheral base address.
- port – Port number to check.
- pstat – Port status from PnSTAT register.

- `error` – Pointer to structure to receive error information.

`static inline uint32_t ESPI_GetPortMsgLen(ESPI_Type *base, uint32_t port)`

Gets the message length from DATAIN register.

This function reads the PnDATAIN register and extracts the DATA_LEN field, which indicates the number of bytes in the current mailbox message. The actual length is DATA_LEN + 1.

Parameters

- `base` – eSPI peripheral base address.
- `port` – Port index.

Returns

Message length in bytes.

`static inline uint32_t ESPI_GetVWire(ESPI_Type *base)`

Reads current Virtual Wire receive status.

Returns raw WIRERO bits so upper layers can inspect incoming VW signals.

Parameters

- `base` – eSPI peripheral base address.

Returns

Raw WIRERO bitfield value.

`status_t ESPI_SendVWire(ESPI_Type *base, espi_vw_wr_flags_t flag, uint32_t value)`

Sends a virtual wire message.

Parameters

- `base` – eSPI peripheral base address.
- `flag` – Virtual wire flag to send, see `espi_vw_wr_flags_t`.
- `value` – Value to set for the wire.

Return values

- `kStatus_Success` – Operation completed.
- `kStatus_Busy` – Previous write still pending.

`void ESPI_TriggerOOBMsg(ESPI_Type *base, espi_handle_t *handle, uint32_t length)`

Triggers an OOB message.

Programs PnOMFLEN.LEN and sets PnIRULESTAT.SSTCL to start an outbound OOB message.

Parameters

- `base` – eSPI peripheral base address.
- `handle` – eSPI handle (provides OOB port index and ACK flag).
- `length` – Message length in bytes (> 0).

`status_t ESPI_SendOOB(ESPI_Type *base, espi_handle_t *handle, const uint8_t *data, uint32_t length, bool announce)`

Send an Out-of-Band (OOB) message.

Copies `length` bytes into the OOB mailbox, triggers the OOB message.

Parameters

- `base` – eSPI peripheral base address.
- `handle` – Optional eSPI handle to arm IRQ-driven ACK handling.

- `data` – Pointer to data to send.
- `length` – Number of bytes to send (>0).
- `announce` – If true, set SSTCL to “Started by MCU” before copy.

Return values

- `kStatus_Success` – Message accepted and triggered.
- `kStatus_InvalidArgument` – Invalid argument.

`status_t` ESPI_ReadOOB(`ESPI_Type` *`base`, `espi_handle_t` *`handle`, `uint8_t` *`buffer`, `uint32_t` *`len`)

Reads the OOB mailbox into a destination buffer.

Copies the OOB message into `buffer` (up to *`len`). *`len` receives the actual number of bytes copied. Returns `kStatus_ESPI_Error` if truncation occurred.

Parameters

- `base` – eSPI peripheral base address.
- `handle` – eSPI handle.
- `buffer` – Destination data buffer.
- `len` – [in] Length of bytes to receive, [out] Length of actual received bytes.

Return values

- `kStatus_Success` – Read successfully (no truncation).
- `kStatus_ESPI_Error` – Message truncated.

static inline void ESPI_SetFlashOpLen(`ESPI_Type` *`base`, `uint32_t` `port`, `uint32_t` `type`, `uint32_t` `length`)

Sets the flash operation length.

Configures the transaction type and payload length for a flash operation.

Parameters

- `base` – eSPI peripheral base address.
- `port` – Hardware port index.
- `type` – Flash transaction type.
- `length` – Payload length in bytes.

void ESPI_SetFlashCompletion(`ESPI_Type` *`base`, `uint32_t` `port`, `uint32_t` `tag`, `uint32_t` `state`, `uint32_t` `type`)

Sets the flash completion status.

Updates the completion status fields for the specified port and tag.

Parameters

- `base` – eSPI peripheral base address.
- `port` – Hardware port index.
- `tag` – Transaction tag.
- `state` – Transaction state.
- `type` – Completion type.

void ESPI_CreateHandle(`ESPI_Type` *`base`, `espi_handle_t` *`handle`, const `espi_config_t` *`config`, `espi_callback_config_t` *`callback`, void *`userData`)

Initializes the eSPI handle.

Parameters

- base – eSPI peripheral base address.
- handle – Pointer to the eSPI handle.
- config – Pointer to the configuration structure.
- callback – Pointer to callback configuration structure.
- userData – User data.

```
void ESPI_FlashCreateHandle(ESPI_Type *base, espi_handle_t *handle, espi_flash_ops_t
                           flashOps, uint32_t flashSize)
```

Initializes the eSPI flash functionality in the handle.

Parameters

- base – eSPI peripheral base address.
- handle – Pointer to the eSPI handle.
- flashOps – Pointer to flash backend operations.
- flashSize – Flash address space size.

```
void ESPI_TransferHandleIRQ(ESPI_Type *base, espi_handle_t *handle)
    eSPI IRQ handle function.
```

This function handles the eSPI transmit and receive IRQ request.

Parameters

- base – eSPI peripheral base address.
- handle – Pointer to the eSPI handle.

```
struct __espi_port_config
```

#include <fsl_espi.h> eSPI per-port configuration structure (PnCFG/PnADDR/PnRAMUSE).

This describes one hardware port (Pn). ESPI_Init will program CFG, RAMUSE and ADDR for each entry if provided in *espi_config_t::portConfig*.

Public Members

uint8_t type

Port type. Refer to *espi_port_type_t*.

uint8_t direction

PnCFG.DIRECTION (for single-direction mailbox).

uint32_t ramOffset

PnRAMUSE.OFF (byte address offset from RAMBASE).

espi_ram_size_t ramSize

PnRAMUSE.LEN encoding (4,8,...,512 bytes).

uint32_t addrOffset

PnADDR.OFF (offset within selected base).

uint8_t addrBase

PnADDR.BASE/ASZ encoding.

uint8_t idxOffset

PnADDR.IDXOFF.

```
struct __espi_config
```

#include <fsl_espi.h> eSPI configuration structure.

Public Members

`uint32_t ramBaseAddr`
RAM base address.

`uint16_t base0Addr`
Base 0 address.

`uint16_t base1Addr`
Base 1 address.

`bool enableSAF`
Enable Slave Attached Flash.

`bool enableOOB`
Enable Out-Of-Band channel.

`bool enableP80`
Enable Port 80 capture/logic.

`bool enableAlertPin`
Alert pin enable.

`uint8_t spiMode`
SPI mode.

`uint8_t busSpeed`
Maximum SPI speed.

`uint8_t enableMode`
See `espi_enable_mode_t`.

`uint8_t SAFEraseSize`
SAF minimum erase sector size, see `espi_saf_erase_size_t`.

`uint8_t maxSAFRxReqSize`
Maximum read request size.

`uint8_t maxPayloadSize`
Maximum payload size.

`uint8_t maxFlashPayloadSize`
Maximum flash payload size.

`uint8_t maxOOBPayloadSize`
Maximum OOB payload size.

`const espi_port_config_t *portConfig`
Port configurations(array).

`uint32_t portCount`
Number of entries in `portConfig`.

`struct _espi_port80_status`
#include <fsl_espi.h> Port 80 status structure.

Public Members

`uint8_t currentCode`
Current POST code.

uint8_t previousCode
Previous POST code.

uint8_t counter
POST code counter (0-15, wraps).

struct __espi_flash_request
#include <fsl_espi.h> eSPI flash request structure.

Public Members

uint32_t addr
Flash address.

uint32_t length
Request length.

uint8_t tag
Transaction tag.

espi_flash_trans_type_t type
Transaction type.

uint8_t *data
Pointer to payload data.

bool readStart
Read request with address and length information.

struct __espi_callback_config
#include <fsl_espi.h> eSPI callback configuration structure.

Public Members

espi_common_callback_t commonCallback
Global event callback.

espi_port_callback_t portCallback
Port event callback.

struct __espi_handle
#include <fsl_espi.h> eSPI handle structure.

Public Members

espi_callback_config_t callback
Callback function.

void *userData
User data.

espi_flash_ops_t flashOps
Flash backend operations.

uint32_t flashSize
Flash address space size for bounds checking.

uint32_t addrOffset[ESPI_PORT_COUNT]
Cached PnADDR.OFF values for each port.

uint8_t oobPort

Port index for OOB (TYPE=Mailbox OOB Split).

uint8_t safPort

Port index for Slave Flash.

uint8_t mafPort

Port index for Bus Master Mem Single.

2.13 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const *ewm_config_t* *config)

Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

[ewm_config_t](#)

Parameters

- `config` – Pointer to the EWM configuration structure.

`static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)`

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- `base` – EWM peripheral base address
- `mask` – The interrupts to enable The parameter can be combination of the following source if defined
 - `kEWM_InterruptEnable`

`static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)`

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- `base` – EWM peripheral base address
- `mask` – The interrupts to disable The parameter can be combination of the following source if defined
 - `kEWM_InterruptEnable`

`static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)`

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

`_ewm_status_flags_t`

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- `base` – EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

`void EWM_Refresh(EWM_Type *base)`

Services the EWM.

This function resets the EWM counter to zero.

Parameters

- `base` – EWM peripheral base address

`FSL_EWM_DRIVER_VERSION`

EWM driver version 2.0.4.

enum `_ewm_interrupt_enable_t`

EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator `kEWM_InterruptEnable`

Enable the EWM to generate an interrupt

enum `_ewm_status_flags_t`

EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator `kEWM_RunningFlag`

Running flag, set when EWM is enabled

typedef struct `_ewm_config` `ewm_config_t`

Describes EWM clock source.

Data structure for EWM configuration.

This structure is used to configure the EWM.

struct `_ewm_config`

#include <fsl_ewm.h> Describes EWM clock source.

Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

bool `enableEwm`

Enable EWM module

bool `enableEwmInput`

Enable EWM_in input

bool `setInputAssertLogic`

EWM_in signal assertion state

bool `enableInterrupt`

Enable EWM interrupt

uint8_t `compareLowValue`

Compare low-register value

uint8_t `compareHighValue`

Compare high-register value

2.14 FGPIO Driver

2.15 FLASH Driver

enum _flash_driver_version_constants

Flash driver version for ROM.

Values:

enumerator kFLASH_DriverVersionName

Flash driver version name.

enumerator kFLASH_DriverVersionMajor

Major flash driver version.

enumerator kFLASH_DriverVersionMinor

Minor flash driver version.

enumerator kFLASH_DriverVersionBugfix

Bugfix for flash driver version.

FSL_ROMAPI_FLASH_DRIVER_VERSION

ROMAPI_FLASH driver version 2.0.0.

Flash driver status codes.

Values:

enumerator kStatus_FLASH_Success

API is executed successfully

enumerator kStatus_FLASH_InvalidArgument

Invalid argument

enumerator kStatus_FLASH_SizeError

Error size

enumerator kStatus_FLASH_AlignmentError

Parameter is not aligned with the specified baseline

enumerator kStatus_FLASH_AddressError

Address is out of range

enumerator kStatus_FLASH_AccessError

Invalid instruction codes and out-of bound addresses

enumerator kStatus_FLASH_ProtectionViolation

The program/erase operation is requested to execute on protected areas

enumerator kStatus_FLASH_CommandFailure

Run-time error during command execution.

enumerator kStatus_FLASH_UnknownProperty

Unknown property.

enumerator kStatus_FLASH_EraseKeyError

API erase key is invalid.

enumerator kStatus_FLASH_RegionExecuteOnly

The current region is execute-only.

enumerator kStatus_FLASH_ExecuteInRamFunctionNotReady

Execute-in-RAM function is not available.

enumerator kStatus_FLASH_CommandNotSupported

Flash API is not supported.

enumerator kStatus_FLASH_ReadOnlyProperty

The flash property is read-only.

enumerator kStatus_FLASH_InvalidPropertyValue

The flash property value is out of range.

enumerator kStatus_FLASH_InvalidSpeculationOption

The option of flash prefetch speculation is invalid.

enumerator kStatus_FLASH_EccError

A correctable or uncorrectable error during command execution.

enumerator kStatus_FLASH_CompareError

Destination and source memory contents do not match.

enumerator kStatus_FLASH_RegulationLoss

A loss of regulation during read.

enumerator kStatus_FLASH_InvalidWaitStateCycles

The wait state cycle set to r/w mode is invalid.

enumerator kStatus_FLASH_OutOfDateCfpaPage

CFPA page version is out of date.

enumerator kStatus_FLASH_BlankIfrPageData

Blank page cannot be read.

enumerator kStatus_FLASH_EncryptedRegionsEraseNotDoneAtOnce

Encrypted flash subregions are not erased at once.

enumerator kStatus_FLASH_ProgramVerificationNotAllowed

Program verification is not allowed when the encryption is enabled.

enumerator kStatus_FLASH_HashCheckError

Hash check of page data is failed.

enumerator kStatus_FLASH_SealedFfrRegion

The FFR region is sealed.

enumerator kStatus_FLASH_FfrRegionWriteBroken

The FFR Spec region is not allowed to be written discontinuously.

enumerator kStatus_FLASH_NmpaAccessNotAllowed

The NMPA region is not allowed to be read/written/erased.

enumerator kStatus_FLASH_CmpaCfgDirectEraseNotAllowed

The CMPA Cfg region is not allowed to be erased directly.

enumerator kStatus_FLASH_FfrBankIsLocked

The FFR bank region is locked.

enumerator kStatus_FLASH_CfpaScratchPageInvalid

CFPA Scratch Page is invalid

enumerator kStatus_FLASH_CfpaVersionRollbackDisallowed

CFPA version rollback is not allowed

enumerator kStatus_FLASH_ReadHidingAreaDisallowed

Flash hiding read is not allowed

enumerator kStatus_FLASH_ModifyProtectedAreaDisallowed

Flash firewall page locked erase and program are not allowed

enumerator kStatus_FLASH_CommandOperationInProgress

The flash state is busy, indicate that a flash command in progress.

kStatusGroupGeneric

Flash driver status group.

kStatusGroupFlashDriver

MAKE_STATUS(group, code)

Constructs a status code value from a group and a code number.

enum _flash_driver_api_keys

Enumeration for Flash driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFLASH_ApiEraseKey

Key value used to validate all flash erase APIs.

FOUR_CHAR_CODE(a, b, c, d)

Constructs the four character code for the Flash driver API key.

Flash Pad Definitions.

Values:

enumerator kSerialFlash_1Pad

enumerator kSerialFlash_2Pads

enumerator kSerialFlash_4Pads

enumerator kSerialFlash_8Pads

FLEXSPI clock configuration type.

Values:

enumerator kFLEXSPIClk_SDR

Clock configure for SDR mode

enumerator kFLEXSPIClk_DDR

Clock configurat for DDR mode

enum _flexspi_read_sample_clk

FLEXSPI Read Sample Clock Source definition.

Values:

enumerator kFLEXSPIReadSampleClk_LoopbackInternally

enumerator kFLEXSPIReadSampleClk_LoopbackFromDqsPad

enumerator kFLEXSPIReadSampleClk_LoopbackFromSckPad

enumerator kFLEXSPIReadSampleClk_ExternalInputFromDqsPad

Flash Type Definition.

Values:

enumerator kFLEXSPIDeviceType_SerialNOR
Flash device is Serial NOR

Flash Configuration Command Type.

Values:

enumerator kDeviceConfigCmdType_Generic
Generic command, for example: configure dummy cycles, drive strength, etc

enumerator kDeviceConfigCmdType_QuadEnable
Quad Enable command

enumerator kDeviceConfigCmdType_Spi2Xpi
Switch from SPI to DPI/QPI/OPI mode

enumerator kDeviceConfigCmdType_Xpi2Spi
Switch from DPI/QPI/OPI to SPI mode

enumerator kDeviceConfigCmdType_Spi2NoCmd
Switch to 0-4-4/0-8-8 mode

enumerator kDeviceConfigCmdType_Reset
Reset device command

enum _flexspi_serial_clk_freq

Defintions for FLEXSPI Serial Clock Frequency.

Values:

enumerator kFLEXSPISerialClk_NoChange

enumerator kFLEXSPISerialClk_30MHz

enumerator kFLEXSPISerialClk_50MHz

enumerator kFLEXSPISerialClk_60MHz

enumerator kFLEXSPISerialClk_75MHz

enumerator kFLEXSPISerialClk_80MHz

enumerator kFLEXSPISerialClk_100MHz

enumerator kFLEXSPISerialClk_133MHz

enumerator kFLEXSPISerialClk_166MHz

Misc feature bit definitions.

Values:

enumerator kFLEXSPIMiscOffset_DiffClkEnable
Bit for Differential clock enable

enumerator kFLEXSPIMiscOffset_Ck2Enable
Bit for CK2 enable

enumerator kFLEXSPIMiscOffset__ParallelEnable

Bit for Parallel mode enable

enumerator kFLEXSPIMiscOffset__WordAddressableEnable

Bit for Word Addressable enable

enumerator kFLEXSPIMiscOffset__SafeConfigFreqEnable

Bit for Safe Configuration Frequency enable

enumerator kFLEXSPIMiscOffset__PadSettingOverrideEnable

Bit for Pad setting override enable

enumerator kFLEXSPIMiscOffset__DdrModeEnable

Bit for DDR clock configuration indication.

enumerator kFLEXSPIMiscOffset__UseValidTimeForAllFreq

Bit for DLLCR settings under all modes

status_t FLASH_Init(*flash_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.

status_t FLASH_EraseSector(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. NOTE: The start address need to be 4 Bytes-aligned.
- lengthInBytes – The length, given in bytes need be 4 Bytes-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – The parameter is not aligned with the specified baseline.

- `kStatus_FLASH_AddressError` – The address is out of range.
- `kStatus_FLASH_EraseKeyError` – The API erase key is invalid.
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` – Flash firewall page locked erase and program are not allowed
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.

`status_t` FLASH_ProgramPhrase(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` – Flash firewall page locked erase and program are not allowed
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t` FLASH_ProgramPage(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash page with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be programmed. Must be word-aligned.
- `src` – A pointer to the source buffer of data that is to be programmed into the flash.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with the specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FLASH_ModifyProtectedAreaDisallowed` – Flash firewall page locked erase and program are not allowed
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

status_t FLASH_VerifyErasePhrase(*flash_config_t* **config*, *uint32_t* *start*, *uint32_t* *lengthInBytes*)

Verifies an erasure of the desired flash area at phrase level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.

- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t FLASH_VerifyErasePage(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)`

Verifies an erasure of the desired flash area at page level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t FLASH_VerifyEraseSector(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)`

Verifies an erasure of the desired flash area at sector level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.

- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t FLASH_VerifyProgram(flash_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, uint32_t *failedAddress, uint32_t *failedData)`

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired flash memory to be verified. Must be word-aligned.
- `lengthInBytes` – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- `expectedData` – A pointer to the expected data that is to be verified against.
- `failedAddress` – A pointer to the returned failing address.
- `failedData` – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_AccessError` – Invalid instruction codes and out-of bounds addresses.
- `kStatus_FLASH_ReadHidingAreaDisallowed` – Flash hiding read is not allowed
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t FLASH_GetProperty(flash_config_t *config, flash_property_tag_t whichProperty, uint32_t *value)`

Returns the desired flash property.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `whichProperty` – The desired property from the list of properties in

- enum flash_property_tag_t
- value – A pointer to the value returned for the desired flash property.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_UnknownProperty – An unknown property tag.

status_t FLASH_GetAPIVersion(*flash_config_t* *config)

Gets the flash driver version.

This function returns the version information of the flash driver.

Parameters

- config – A pointer to the storage for the driver runtime state.

Returns

Flash driver version information.

status_t IFR_VerifyErasePhrase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

Verifies an erasure of the desired IFR area at phrase level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the IFR is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired IFR memory to be verified.
- lengthInBytes – The length, given in bytes, to be verified.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FLASH_AddressError – Address is out of range.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.
- kStatus_FLASH_RegulationLoss – A loss of regulation during read.
- kStatus_FLASH_EccError – A correctable or uncorrectable error during command execution.

status_t IFR_VerifyErasePage(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes)

Verifies an erasure of the desired IFR area at page level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the IFR is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired IFR memory to be verified.
- lengthInBytes – The length, given in bytes, to be verified.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t IFR_VerifyEraseSector(flash_config_t *config, uint32_t start, uint32_t lengthInBytes)`

Verifies an erasure of the desired IFR area at sector level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the IFR is erased to the specified read margin level.

Parameters

- `config` – A pointer to the storage for the driver runtime state.
- `start` – The start address of the desired IFR memory to be verified.
- `lengthInBytes` – The length, given in bytes, to be verified.

Return values

- `kStatus_FLASH_Success` – API was executed successfully.
- `kStatus_FLASH_InvalidArgument` – An invalid argument is provided.
- `kStatus_FLASH_AlignmentError` – Parameter is not aligned with specified baseline.
- `kStatus_FLASH_AddressError` – Address is out of range.
- `kStatus_FLASH_CommandFailure` – Run-time error during the command execution.
- `kStatus_FLASH_CommandNotSupported` – Flash API is not supported.
- `kStatus_FLASH_RegulationLoss` – A loss of regulation during read.
- `kStatus_FLASH_EccError` – A correctable or uncorrectable error during command execution.

`status_t LPSPI_EepromInit(void)`

Initialize SPI NOR/EEPROM.

Parameters

- `baudRate` – Config baudrate for SPI.

Return values

- `kStatus_Fail` – EEPROM initialization failed.
- `kStatus_InvalidArgument` – Invalid input parameter.
- `kStatus_ROM_LPSPI_Busy` – LPSPI transfer is busy.
- `kStatus_Success` – The eeprom is initialized successfully.

status_t LPSPI_EepromRead(uint32_t address, uint32_t NoOfBytes, uint8_t *restrict buffer)
Read data via SPI NOR/EEPROM.

Parameters

- dest – A pointer to the buffer of data that is to be read from eeprom.
- length – The length, given in bytes to be read.
- address – The start address of the desired eeprom memory to be read.
- requestFastRead – The type of Read eeprom command. FALSE: Use the general Read command to read eeprom TRUE: Use Fast Read command to read data from eeprom.

Return values

- kStatus_Fail – Failed to read data from eeprom.
- kStatus_Success – Read data from eeprom successfully.
- kStatus_InvalidArgument – Invalid input parameter.
- kStatus_ROM_LPSPI_Busy – LPSPI transfer is busy.

status_t LPSPI_EepromWrite(uint32_t address, uint32_t NoOfBytes, const uint8_t *buffer)
Write data via SPI NOR/EEPROM.

Parameters

- data – A pointer to the source buffer of data that is to be programmed into the eeprom.
- length – The length, given in bytes to be programmed.
- address – The start address of the desired eeprom memory to be programmed.

Return values

- kStatus_Fail – Failed to write data to eeprom.
- kStatus_Success – Successfully write data to eeprom.
- kStatus_InvalidArgument – Invalid input parameter.
- kStatus_ROM_LPSPI_Busy – LPSPI transfer is busy.

status_t LPSPI_EepromErase(uint32_t address, uint32_t length)
Erase data via SPI NOR/EEPROM.

Parameters

- address – The start address of the desired eeprom memory to be erased.
- option – The length, given in bytes to be erased.

Return values

- kStatus_Fail – Failed to erase data from the eeprom.
- kStatus_Success – Erase data from eeprom successfully.
- kStatus_InvalidArgument – Invalid input parameter.
- kStatus_ROM_LPSPI_Busy – LPSPI transfer is busy.

status_t LPSPI_EepromConfig(uint32_t *config)
Configure SPI EEPROM.

This function configures the SPI EEPROM with the given configuration parameters.

Parameters

- `config` – A pointer to the configuration parameters. The specific format depends on the EEPROM device type.

Return values

- `kStatus_Fail` – Failed to erase data from the eeprom.
- `kStatus_Success` – Erase data from eeprom successfully.
- `kStatus_InvalidArgument` – Invalid input parameter.
- `kStatus_ROM_LPSPI_Busy` – LPSPI transfer is busy.

`status_t` LPSPI_EepromFlush(void)

Flush SPI EEPROM.

This function flushes any pending operations and ensures all data is written to the SPI EEPROM.

Return values

- `kStatus_Fail` – Failed to erase data from the eeprom.
- `kStatus_Success` – Erase data from eeprom successfully.
- `kStatus_InvalidArgument` – Invalid input parameter.
- `kStatus_ROM_LPSPI_Busy` – LPSPI transfer is busy.

`status_t` LPSPI_EepromEraseAll(void)

Erase all SPI EEPROM.

This function erases the entire SPI EEPROM device.

Return values

- `kStatus_Fail` – Failed to erase data from the eeprom.
- `kStatus_Success` – Erase data from eeprom successfully.
- `kStatus_InvalidArgument` – Invalid input parameter.
- `kStatus_ROM_LPSPI_Busy` – LPSPI transfer is busy.

`MAKE_VERSION(major, minor, bugfix)`

Constructs the version number for drivers.

`enum _flash_property_tag`

Enumeration for various flash properties.

Values:

enumerator `kFLASH_PropertyPflashSectorSize`

Pflash sector size property.

enumerator `kFLASH_PropertyPflashTotalSize`

Pflash total size property.

enumerator `kFLASH_PropertyPflashBlockSize`

Pflash block size property.

enumerator `kFLASH_PropertyPflashBlockCount`

Pflash block count property.

enumerator `kFLASH_PropertyPflashBlockBaseAddr`

Pflash block base address property.

enumerator `kFLASH_PropertyPflashPageSize`

Pflash page size property.

enumerator kFLASH_PropertyPflashSystemFreq
System Frequency System Frequency.

enumerator kFLASH_PropertyFfrSectorSize
FFR sector size property.

enumerator kFLASH_PropertyFfrTotalSize
FFR total size property.

enumerator kFLASH_PropertyFfrBlockBaseAddr
FFR block base address property.

enumerator kFLASH_PropertyFfrPageSize
FFR page size property.

enum _flash_max_erase_page_value
Enumeration for flash max pages to erase.

Values:

enumerator kFLASH_MaxPagesToErase
The max value in pages to erase.

enum _flash_alignment_property
Enumeration for flash alignment property.

Values:

enumerator kFLASH_AlignementUnitVerifyErase
The alignment unit in bytes used for verify erase operation.

enumerator kFLASH_AlignementUnitProgram
The alignment unit in bytes used for program operation.

enumerator kFLASH_AlignementUnitSingleWordRead
The alignment unit in bytes used for verify program operation. The alignment unit in bytes used for SingleWordRead command.

enum _flash_read_ecc_option
Enumeration for flash read ecc option.

Values:

enumerator kFLASH_ReadWithEccOn

enumerator kFLASH_ReadWithEccOff
ECC is on

enum _flash_read_margin_option
Enumeration for flash read margin option.

Values:

enumerator kFLASH_ReadMarginNormal
Normal read

enumerator kFLASH_ReadMarginVsProgram
Margin vs. program

enumerator kFLASH_ReadMarginVsErase
Margin vs. erase

enumerator kFLASH_ReadMarginIllegalBitCombination
Illegal bit combination

enum _flash_read_dmacc_option

Enumeration for flash read dmacc option.

Values:

enumerator kFLASH_ReadDmaccDisabled

Memory word

enumerator kFLASH_ReadDmaccEnabled

DMACC word

enum _flash_ramp_control_option

Enumeration for flash ramp control option.

Values:

enumerator kFLASH_RampControlDivisionFactorReserved

Reserved

enumerator kFLASH_RampControlDivisionFactor256

$\text{clk48mhz} / 256 = 187.5\text{KHz}$

enumerator kFLASH_RampControlDivisionFactor128

$\text{clk48mhz} / 128 = 375\text{KHz}$

enumerator kFLASH_RampControlDivisionFactor64

$\text{clk48mhz} / 64 = 750\text{KHz}$

enum _flexspi_status_groups

FLEXSPI status group numbers.

Values:

enumerator kStatusROMGroup_FLEXSPI

Group number for ROM FLEXSPI status codes.

enumerator kStatusROMGroup_FLEXSPINOR

ROM FLEXSPI NOR status group number.

enum _flexspi_nor_status

FLEXSPI NOR status.

Values:

enumerator kStatus_FLEXSPINOR_ProgramFail

Status for Page programming failure

enumerator kStatus_FLEXSPINOR_EraseSectorFail

Status for Sector Erase failure

enumerator kStatus_FLEXSPINOR_EraseAllFail

Status for Chip Erase failure

enumerator kStatus_FLEXSPINOR_WaitTimeout

Status for timeout

enumerator kStatus_FlexSPINOR_NotSupported

Status for PageSize overflow

enumerator kStatus_FlexSPINOR_WriteAlignmentError

Status for Alignement error

enumerator kStatus_FlexSPINOR_CommandFailure

Status for Erase/Program Verify Error

enumerator kStatus_FlexSPINOR_SFDP_NotFound

Status for SFDP read failure

enumerator kStatus_FLEXSPINOR_Unsupported_SFDP_Version

Status for Unrecognized SFDP version

enumerator kStatus_FLEXSPINOR_Flash_NotFound

Status for Flash detection failure

enumerator kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed

Status for DDR Read dummy probe failure

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Status for Sequence Execution timeout

enumerator kStatus_FLEXSPI_InvalidSequence

Status for Invalid Sequence

enumerator kStatus_FLEXSPI_DeviceTimeout

Status for Device timeout

Configure the device_type of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorCfgOption_Tag

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_SDR

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_DDR

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH1V8

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH3V0

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalSDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalSDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalDDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalSDR

Configure the quad_mode_setting of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorQuadMode_NotConfig

enumerator kSerialNorQuadMode_StatusReg1_Bit6

enumerator kSerialNorQuadMode_StatusReg2_Bit1

enumerator kSerialNorQuadMode_StatusReg2_Bit7

enumerator kSerialNorQuadMode_StatusReg2_Bit1_0x31

FLEXSPI NOR Octal mode.

Values:

enumerator kSerialNorOctalMode_NoOctalEnableBit

enumerator kSerialNorOctalMode_HasOctalEnableBit

miscellaneous mode

Values:

enumerator kSerialNorEnhanceMode_Disabled

enumerator kSerialNorEnhanceMode_0_4_4_Mode

enumerator kSerialNorEnhanceMode_0_8_8_Mode

enumerator kSerialNorEnhanceMode_DataOrderSwapped

enumerator kSerialNorEnhanceMode_2ndPinMux

enumerator kSerialNorEnhanceMode_InternalLoopback

enumerator kSerialNorEnhanceMode_SpiMode

enumerator kSerialNorEnhanceMode_ExtDqs

FLEXSPI NOR reset logic options.

Values:

enumerator kFlashResetLogic_Disabled

enumerator kFlashResetLogic_ResetPin

enumerator kFlashResetLogic_JedecHwReset

Configure the flash_connection of “serial_nor_config_option_t” structure.

Values:

enumerator kSerialNorConnection_SinglePortA

enumerator kSerialNorConnection_Parallel

enumerator kSerialNorConnection_SinglePortB

enumerator kSerialNorConnection_BothPorts

FLEXSPI ROOT clock source related definitions.

Values:

enumerator kFLEXSPIClkSrc_MainClk

enumerator kFLEXSPIClkSrc_Pll0

enumerator kFLEXSPIClkSrc_FroHf

enumerator kFLEXSPIClkSrc_Pll1

Restore sequence options Configure the restore_sequence of “flash_run_context_t” structure.

Values:

enumerator kRestoreSequence__None
enumerator kRestoreSequence__HW__Reset
enumerator kRestoreSequence__QPI_4__0xFFs
enumerator kRestoreSequence__QPI_Mode__0x00
enumerator kRestoreSequence__8QPI_FF
enumerator kRestoreSequence__Send_F0
enumerator kRestoreSequence__Send_66_99
enumerator kRestoreSequence__Send_6699_9966
enumerator kRestoreSequence__Send_06_FF
 Adesto EcoXIP
enumerator kRestoreSequence__QPI_5__0xFFs
enumerator kRestoreSequence__Send_QPI_8__0xFFs
enumerator kRestoreSequence__Wakeup__0xAB
enumerator kRestoreSequence__Wakeup__0xAB_54

Port mode options.

Values:

enumerator kFlashInstMode__ExtendedSpi
enumerator kFlashInstMode__0_4_4__SDR
enumerator kFlashInstMode__0_4_4__DDR
enumerator kFlashInstMode__DPI__SDR
enumerator kFlashInstMode__DPI__DDR
enumerator kFlashInstMode__QPI__SDR
enumerator kFlashInstMode__QPI__DDR
enumerator kFlashInstMode__OPI__SDR
enumerator kFlashInstMode__OPI__DDR

Manufacturer ID.

Values:

enumerator kSerialFlash__ISSI__ManufacturerID
 Manufacturer ID of the ISSI serial flash
enumerator kSerialFlash__Adesto__ManufacturerID
 Manufacturer ID of the Adesto Technologies serial flash

enumerator kSerialFlash_Winbond_ManufacturerID
Manufacturer ID of the Winbond serial flash

enumerator kSerialFlash_Cypress_ManufacturerID
Manufacturer ID for Cypress

enum _flexspi_operation
flexspi operation.

Values:

enumerator kFLEXSPIOperation_Command
Only command, both TX and RX buffer are ignored.

enumerator kFLEXSPIOperation_Config
Configure device mode, the TX FIFO size is fixed in LUT.

enumerator kFLEXSPIOperation_Write
Write, only TX buffer is effective

enumerator kFLEXSPIOperation_Read
Read, only Rx Buffer is effective.

enumerator kFLEXSPIOperation_End

enum flexspi_clock_type_t
FLEXSPI Clock Type.

Values:

enumerator kFlexSpiClock_CoreClock
ARM Core Clock

enumerator kFlexSpiClock_AhbClock
AHB clock

enumerator kFlexSpiClock_SerialRootClock
Serial Root Clock

enumerator kFlexSpiClock_IpgClock
IPG clock

LPSPFI Flash status codes.

Values:

enumerator kStatus_ROM_LPSPFI_Busy
LPSPFI transfer is busy.

enumerator kStatus_ROM_LPSPFI_Error
LPSPFI driver error.

enumerator kStatus_ROM_LPSPFI_Idle
LPSPFI is idle.

enumerator kStatus_ROM_LPSPFI_OutOfRange
LPSPFI transfer out Of range.

typedef enum _flash_property_tag flash_property_tag_t
Enumeration for various flash properties.

typedef struct _flash_ecc_log flash_ecc_log_t
Flash ECC log info.

```
typedef struct _flash_mode_config flash_mode_config_t
```

Flash controller paramter config.

```
typedef struct _flash_ffr_config flash_ffr_config_t
```

Flash controller paramter config.

```
typedef union StandardVersion standard_version_t
```

Structure of version property.

```
typedef struct _serial_nor_config_option serial_nor_config_option_t
```

Serial NOR configuration option.

```
typedef struct _lut_sequence flexspi_lut_seq_t
```

FLEXSPI LUT Sequence structure.

```
typedef struct _FlexSPIConfig flexspi_mem_config_t
```

FLEXSPI Memory Configuration Block.

```
typedef struct _flexspi_nor_config flexspi_nor_config_t
```

Serial NOR configuration block.

```
typedef enum _flexspi_operation flexspi_operation_t
```

flexspi operation.

```
typedef struct _flexspi_xfer flexspi_xfer_t
```

FLEXSPI Transfer Context.

```
status_t FLEXSPI_NorFlash_Init(uint32_t instance, flexspi_nor_config_t *config)
```

Initialize Serial NOR devices via FLEXSPI.

This function checks and initializes the FLEXSPI module for the other FLEXSPI APIs.

Parameters

- *instance* – storage the instance of FLEXSPI.
- *config* – A pointer to the storage for the driver runtime state.

Return values

- *kStatus_Success* – Api was executed succesfully.
- *kStatus_InvalidArgument* – A invalid argument is provided.
- *kStatus_FLEXSPI_InvalidSequence* – A invalid Sequence is provided.
- *kStatus_FLEXSPI_SequenceExecutionTimeout* – Sequence Execution time-out.
- *kStatus_FLEXSPI_DeviceTimeout* – the device timeout

```
status_t FLEXSPI_NorFlash_ProgramPage(uint32_t instance, flexspi_nor_config_t *config,  
uint32_t dstAddr, const uint32_t *src)
```

Program page data to Serial NOR via FLEXSPI.

This function programs the NOR flash memory with the dest address for a given flash area as determined by the dst address and the length.

Parameters

- *instance* – storage the instance of FLEXSPI.
- *config* – A pointer to the storage for the driver runtime state.
- *dstAddr* – A pointer to the desired flash memory to be programmed. NOTE: It is recommended that use page aligned access; If the *dst_addr* is not aligned to page,the driver automatically aligns address down with the page address.

- `src` – A pointer to the source buffer of data that is to be programmed into the NOR flash.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t FLEXSPI_NorFlash_EraseAll(uint32_t instance, flexspi_nor_config_t *config)`

Erase all the Serial NOR devices connected on FLEXSPI.

Parameters

- `instance` – storage the instance of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

`status_t FLEXSPI_NorFlash_EraseSector(uint32_t instance, flexspi_nor_config_t *config, uint32_t address)`

Erase one sector specified by address.

This function erases one of NOR flash sectors based on the desired address.

Parameters

- `instance` – storage the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `address` – The start address of the desired NOR flash memory to be erased.
NOTE: It is recommended that use sector-aligned access nor device; If `dstAddr` is not aligned with the sector, The driver automatically aligns address down with the sector address.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.
- `kStatus_FLEXSPI_DeviceTimeout` – the device timeout

status_t FLEXSPI_NorFlash_EraseBlock(*uint32_t* instance, *flexspi_nor_config_t* *config, *uint32_t* address)

Erase one block specified by address.

This function erases one block of NOR flash based on the desired address.

Parameters

- instance – storage the index of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- address – The start address of the desired NOR flash memory to be erased. NOTE: It is recommended that use block-aligned access nor device; If dstAddr is not aligned with the block, The driver automatically aligns address down with the block address.

Return values

- kStatus_Success – Api was executed successfully.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_GetConfig(*uint32_t* instance, *flexspi_nor_config_t* *config, *serial_nor_config_option_t* *option)

Get FLEXSPI NOR Configuration Block based on specified option.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- option – A pointer to the storage Serial NOR Configuration Option Context.

Return values

- kStatus_Success – Api was executed successfully.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_Erase(*uint32_t* instance, *flexspi_nor_config_t* *config, *uint32_t* start, *uint32_t* length)

Erase Flash Region specified by address and length.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- instance – storage the index of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired NOR flash memory to be erased. NOTE: It is recommended that use sector-aligned access nor device; If

dstAddr is not aligned with the sector, the driver automatically aligns address down with the sector address.

- length – The length, given in bytes to be erased. NOTE: It is recommended that use sector-aligned access nor device; If length is not aligned with the sector, the driver automatically aligns up with the sector.

Return values

- kStatus__Success – Api was executed successfully.
- kStatus__InvalidArgument – A invalid argument is provided.
- kStatus__FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus__FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus__FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_Read(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t *dst, uint32_t start, uint32_t bytes)

Read data from Serial NOR via FLEXSPI.

This function read the NOR flash memory with the start address for a given flash area as determined by the dst address and the length.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- dst – A pointer to the dest buffer of data that is to be read from the NOR flash. NOTE: It is recommended that use page aligned access; If the dstAddr is not aligned to page, the driver automatically aligns address down with the page address.
- start – The start address of the desired NOR flash memory to be read.
- bytes – The length, given in bytes to be read.

Return values

- kStatus__Success – Api was executed successfully.
- kStatus__InvalidArgument – A invalid argument is provided.
- kStatus__FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus__FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus__FLEXSPI_DeviceTimeout – the device timeout

status_t FLEXSPI_NorFlash_CommandXfer(uint32_t instance, *flexspi_xfer_t* *xfer)
FLEXSPI command.

This function is used to perform the command write sequence to the NOR device.

Parameters

- instance – storage the index of FLEXSPI.
- xfer – A pointer to the storage FLEXSPI Transfer Context.

Return values

- kStatus__Success – Api was executed successfully.
- kStatus__InvalidArgument – A invalid argument is provided.
- kStatus__ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.

- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.

`status_t FLEXSPI_NorFlash_UpdateLut(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t numberOfSeq)`

Configure FLEXSPI Lookup table.

Parameters

- `instance` – storage the index of FLEXSPI.
- `seqIndex` – storage the sequence Id.
- `lutBase` – A pointer to the look-up-table for command sequences.
- `numberOfSeq` – storage sequence number.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.

`status_t FLEXSPI_NorFlash_SetClockSource(uint32_t clockSource)`

Set the clock source for FLEXSPI NOR.

Parameters

- `clockSource` – Clock source for FLEXSPI NOR. See to “`flexspi_nor_clock_source`”.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.

`void FLEXSPI_NorFlash_ConfigClock(uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode)`

config flexspi clock

Parameters

- `instance` – storage the index of FLEXSPI.
- `freqOption` – pointer to FlexSPIFlexSPI flash serial clock frequency.
- `sampleClkMode` – pointer to configure the FlexSPI clock configuration type.

`status_t FLEXSPI_NorFlash_PartialProgram(uint32_t instance, flexspi_nor_config_t *config, uint32_t dstAddr, const uint32_t *src, uint32_t length)`

Partial program data to FlexSPI NOR flash.

Parameters

- `instance` – storage the index of FLEXSPI.
- `config` – A pointer to the storage for the driver runtime state.
- `dstAddr` – A pointer to the desired flash memory to be programmed.
- `src` – A pointer to the source buffer of data that is to be programmed into the NOR flash.

- `length` – The length, given in bytes (not words or long-words) to be programmed.

Return values

- `kStatus_Success` – Api was executed succesfully.
- `kStatus_InvalidArgument` – A invalid argument is provided.
- `kStatus_ROM_FLEXSPI_InvalidSequence` – A invalid Sequence is provided.
- `kStatus_ROM_FLEXSPI_SequenceExecutionTimeout` – Sequence Execution timeout.

`FLEXSPI_FEATURE_HAS_PARALLEL_MODE`

FLEXSPI Feature related definitions

`FSL_ROM_FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)`

`CMD_SDR`

`CMD_DDR`

`RADDR_SDR`

`RADDR_DDR`

`CADDR_SDR`

`CADDR_DDR`

`MODE1_SDR`

`MODE1_DDR`

`MODE2_SDR`

`MODE2_DDR`

`MODE4_SDR`

`MODE4_DDR`

`MODE8_SDR`

`MODE8_DDR`

`WRITE_SDR`

`WRITE_DDR`

`READ_SDR`

`READ_DDR`

`LEARN_SDR`

`LEARN_DDR`

`DATSZ_SDR`

`DATSZ_DDR`

`DUMMY_SDR`

`DUMMY_DDR`

DUMMY_RWDS_SDR

DUMMY_RWDS_DDR

JMP_ON_CS

FLEXSPI_STOP

FLEXSPI_1PAD

FLEXSPI_2PAD

FLEXSPI_4PAD

FLEXSPI_8PAD

NOR_CMD_LUT_SEQ_IDX_READ

NOR LUT sequence index used for default LUT assignment NOTE: The will take effect if the lut sequences are not customized.

READ LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS

Read Status LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI

Read status DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE

Write Enable sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI

Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_ERASESECTOR

Erase Sector sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READID

NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK

Erase Block sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM

Program sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_CHIPERASE

Chip Erase sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_READ_SFDP

Read SFDP sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD

Restore 0-4/0-8 mode sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD

Exit 0-4/0-8 mode sequence id in lookupTable stored in config blobk

kROM_StatusGroup_LPSPi

ROM Group number for LPSPi status codes.

struct _flash_ecc_log

#include <fsl_flash.h> Flash ECC log info.

```
struct _flash_mode_config
    #include <fsl_flash.h> Flash controller paramter config.
struct _flash_ffr_config
    #include <fsl_flash.h> Flash controller paramter config.
union StandardVersion
    #include <fsl_flash.h> Structure of version property.
```

Public Members

```
struct StandardVersion
    uint32_t version
        combined version numbers
struct flash_config_t
    #include <fsl_flash.h> Flash driver state information.
    An instance of this structure is allocated by the user of the flash driver and passed into each
    of the driver APIs.
```

Public Members

```
uint32_t PFlashBlockBase
    A base address of the first PFlash block
uint32_t PFlashTotalSize
    The size of the combined PFlash block.
uint32_t PFlashBlockCount
    A number of PFlash blocks.
uint32_t PFlashPageSize
    The size in bytes of a page of PFlash.
uint32_t PFlashSectorSize
    The size in bytes of a sector of PFlash.
struct _serial_nor_config_option
    #include <fsl_flexspi_nor_flash.h> Serial NOR configuration option.
union flash_run_context_t
    #include <fsl_flexspi_nor_flash.h>
```

Public Members

```
struct flash_run_context_t B
    uint32_t U
struct _lut_sequence
    #include <fsl_flexspi_nor_flash.h> FLEXSPI LUT Sequence structure.
```

Public Members

uint8_t seqNum
Sequence Number, valid number: 1-16

uint8_t seqId
Sequence Index, valid number: 0-15

struct flexspi_dll_time_t
#include <fsl_flexspi_nor_flash.h>

Public Members

uint8_t time_100ps
Data valid time, in terms of 100ps

uint8_t delay_cells
Data valid time, in terms of delay cells

struct __FlexSPIConfig
#include <fsl_flexspi_nor_flash.h> FLEXSPI Memory Configuration Block.

Public Members

uint32_t tag
[0x000-0x003] Tag, fixed value 0x42464346UL

uint32_t version
[0x004-0x007] Version, [31:24] - 'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix

uint32_t reserved0
[0x008-0x00b] Reserved for future use

uint8_t readSampleClkSrc
[0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3

uint8_t csHoldTime
[0x00d-0x00d] CS hold time, default value: 3

uint8_t csSetupTime
[0x00e-0x00e] CS setup time, default value: 3

uint8_t columnAddressWidth
[0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For Serial NAND, need to refer to datasheet

uint8_t deviceModeCfgEnable
[0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

uint8_t deviceModeType
[0x011-0x011] Specify the configuration command type: Quad Enable, DPI/QPI/OPI switch, Generic configuration, etc.

uint16_t waitTimeCfgCommands
[0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for DPI/QPI/OPI switch or reset command

flexspi_lut_seq_t deviceModeSeq
[0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUT sequence number, [31:16] Reserved

`uint32_t deviceModeArg`
[0x018-0x01b] Argument/Parameter for device configuration

`uint8_t configCmdEnable`
[0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable

`uint8_t configModeType[3]`
[0x01d-0x01f] Configure Mode Type, similar as `deviceModeType`

flexspi_lut_seq_t `configCmdSeqs[3]`
[0x020-0x02b] Sequence info for Device Configuration command, similar as `deviceModeSeq`

`uint32_t reserved1`
[0x02c-0x02f] Reserved for future use

`uint32_t configCmdArgs[3]`
[0x030-0x03b] Arguments/Parameters for device Configuration commands

`uint32_t reserved2`
[0x03c-0x03f] Reserved for future use

`uint32_t controllerMiscOption`
[0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more details

`uint8_t deviceType`
[0x044-0x044] Device Type: See Flash Type Definition for more details

`uint8_t sflashPadType`
[0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal

`uint8_t serialClkFreq`
[0x046-0x046] Serial Flash Frequency, device specific definitions, See System Boot Chapter for more details

`uint8_t lutCustomSeqEnable`
[0x047-0x047] LUT customization Enable, it is required if the program/erase cannot be done using 1 LUT sequence, currently, only applicable to HyperFLASH

`uint32_t reserved3[2]`
[0x048-0x04f] Reserved for future use

`uint32_t sflashA1Size`
[0x050-0x053] Size of Flash connected to A1

`uint32_t sflashA2Size`
[0x054-0x057] Size of Flash connected to A2

`uint32_t sflashB1Size`
[0x058-0x05b] Size of Flash connected to B1

`uint32_t sflashB2Size`
[0x05c-0x05f] Size of Flash connected to B2

`uint32_t csPadSettingOverride`
[0x060-0x063] CS pad setting override value

`uint32_t sclkPadSettingOverride`
[0x064-0x067] SCK pad setting override value

`uint32_t dataPadSettingOverride`
[0x068-0x06b] data pad setting override value

`uint32_t dqsPadSettingOverride`
 [0x06c-0x06f] DQS pad setting override value

`uint32_t timeoutInMs`
 [0x070-0x073] Timeout threshold for read status command

`uint32_t commandInterval`
 [0x074-0x077] CS deselect interval between two commands

flexspi_dll_time_t `dataValidTime[2]`
 [0x078-0x07b] CLK edge to data valid time for PORT A and PORT B

`uint16_t busyOffset`
 [0x07c-0x07d] Busy offset, valid value: 0-31

`uint16_t busyBitPolarity`
 [0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 - busy flag is 0 when flash device is busy

`uint32_t lookupTable[64]`
 [0x080-0x17f] Lookup table holds Flash command sequences

flexspi_lut_seq_t `lutCustomSeq[12]`
 [0x180-0x1af] Customizable LUT Sequences

`uint32_t dll0CrVal`
 [0x1b0-0x1b3] Customizable DLL0CR setting

`uint32_t dll1CrVal`
 [0x1b4-0x1b7] Customizable DLL1CR setting

`uint32_t reserved4[2]`
 [0x1b8-0x1bf] Reserved for future use

`struct _flexspi_nor_config`
 #include <fsl_flexspi_nor_flash.h> Serial NOR configuration block.

Public Members

flexspi_mem_config_t `memConfig`
 Common memory configuration info via FLEXSPI

`uint32_t pageSize`
 Page size of Serial NOR

`uint32_t sectorSize`
 Sector size of Serial NOR

`uint8_t ipcmdSerialClkFreq`
 Clock frequency for IP command

`uint8_t isUniformBlockSize`
 Sector/Block size is the same

`uint8_t isDataOrderSwapped`
 Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)

`uint8_t reserved0[1]`
 Reserved for future use

uint8_t serialNorType
Serial NOR Flash type: 0/1/2/3

uint8_t needExitNoCmdMode
Need to exit NoCmd mode before other IP command

uint8_t halfClkForNonReadCmd
Half the Serial Clock for non-read command: true/false

uint8_t needRestoreNoCmdMode
Need to Restore NoCmd mode after IP commmand execution

uint32_t blockSize
Block size

uint32_t flashStateCtx
Flash State Context

uint32_t reserve2[10]
Reserved for future use

struct __flexspi_xfer
#include <fsl_flexspi_nor_flash.h> FLEXSPI Transfer Context.

Public Members

flexspi_operation_t operation
FLEXSPI operation

uint32_t baseAddress
FLEXSPI operation base address

uint32_t seqId
Sequence Id

uint32_t seqNum
Sequence Number

bool isParallelModeEnable
Is a parallel transfer

uint32_t *txBuffer
Tx buffer

uint32_t txSize
Tx size in bytes

uint32_t *rxBuffer
Rx buffer

uint32_t rxSize
Rx size in bytes

struct readSingleWord

struct setWriteMode

struct setReadMode

struct __unnamed5__

Public Members

uint8_t bugfix
 bugfix version [7:0]
uint8_t minor
 minor version [15:8]
uint8_t major
 major version [23:16]
char name
 name [31:24]

union option0

Public Members

struct *_serial_nor_config_option* B
uint32_t U

struct B

Public Members

uint32_t max_freq
 Maximum supported Frequency
uint32_t misc_mode
 miscellaneous mode
uint32_t quad_mode_setting
 Quad mode setting
uint32_t cmd_pads
 Command pads
uint32_t query_pads
 SFDP read pads
uint32_t device_type
 Device type
uint32_t option_size
 Option size, in terms of uint32_t, size = (option_size + 1) * 4
uint32_t tag
 Tag, must be 0x0E

union option1

Public Members

struct *_serial_nor_config_option* B
uint32_t U

struct B

Public Members

uint32_t dummy__cycles

Dummy cycles before read

uint32_t status__override

Override status register value during device mode configuration

uint32_t pinmux__group

The pinmux group selection

uint32_t dqs__pinmux__group

The DQS Pinmux Group Selection

uint32_t drive__strength

The Drive Strength of FLEXSPI Pads

uint32_t flash__connection

Flash connection option: 0 - Single Flash connected to port A, 1 - Parallel mode, 2 - Single Flash connected to Port B

struct B

2.16 FlexIO: FlexIO Driver

2.17 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

- userConfig – pointer to flexio_config_t structure

void FLEXIO_Init(FLEXIO_Type *base, const *flexio_config_t* *userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_GetDefaultConfig().

Example

```
flexio_config_t config = {
    .enableFlexio = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio_config_t structure

```
void FLEXIO__Deinit(FLEXIO_Type *base)
```

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

Note: After calling this API, call the FLEXIO_Init to use the FlexIO module.

Parameters

- base – FlexIO peripheral base address

```
uint32_t FLEXIO__GetInstance(FLEXIO_Type *base)
```

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

```
void FLEXIO__Reset(FLEXIO_Type *base)
```

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

```
static inline void FLEXIO__Enable(FLEXIO_Type *base, bool enable)
```

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

```
static inline uint32_t FLEXIO__ReadPinInput(FLEXIO_Type *base)
```

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns

FlexIO pin input data

```
static inline uint8_t FLEXIO__GetShifterState(FLEXIO_Type *base)
```

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns

current State pointer

```
void FLEXIO__SetShifterConfig(FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
```

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {  
    .timerSelect = 0,  
    .timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,  
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
```

(continues on next page)

(continued from previous page)

```
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

```
void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t
                          *timerConfig)
```

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFThSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

```
static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index,
                                       flexio_timer_decrement_source_t clocksource)
```

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)
Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)
Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$


```
static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

```
static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)
```

Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

```
static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)
```

Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

```
static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

```
static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)
```

Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

```
static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

```
static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)
```

Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

```
static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)
```

Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by $(1 \ll \text{timer index})$

```
static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)
```

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$
- enable – True to enable, false to disable.

```
uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, flexio_shifter_buffer_type_t type, uint8_t index)
```

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio_shifter_buffer_type_t

- index – Shifter index

Returns

Corresponding shifter buffer index

status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, *flexio_isr_t* isr)

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_UnregisterHandleIRQ(void *base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 0.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 1.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- output – FLEXIO pin output logic level.

- 0: corresponding pin output low-logic level.
- 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)
 Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)
 Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL_FLEXIO_DRIVER_VERSION
 FlexIO driver version.

enum _flexio_timer_trigger_polarity
 Define time of timer trigger polarity.

Values:

enumerator kFLEXIO_TimerTriggerPolarityActiveHigh
 Active high.

enumerator kFLEXIO_TimerTriggerPolarityActiveLow
 Active low.

enum _flexio_timer_trigger_source
 Define type of timer trigger source.

Values:

enumerator kFLEXIO_TimerTriggerSourceExternal
 External trigger selected.

enumerator kFLEXIO_TimerTriggerSourceInternal
 Internal trigger selected.

enum _flexio_pin_config
 Define type of timer/shifter pin configuration.

Values:

enumerator kFLEXIO_PinConfigOutputDisabled
 Pin output disabled.

enumerator kFLEXIO_PinConfigOpenDrainOrBidirection
 Pin open drain or bidirectional output enable.

enumerator kFLEXIO_PinConfigBidirectionOutputData
 Pin bidirectional output data.

enumerator kFLEXIO_PinConfigOutput
 Pin output.

enum _flexio_pin_polarity
 Definition of pin polarity.

Values:

enumerator kFLEXIO__PinActiveHigh
Active high.

enumerator kFLEXIO__PinActiveLow
Active low.

enum _flexio_timer_mode
Define type of timer work mode.

Values:

enumerator kFLEXIO__TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO__TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO__TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO__TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO__TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum _flexio_timer_output
Define type of timer initial output or timer reset condition.

Values:

enumerator kFLEXIO__TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO__TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO__TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO__TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum _flexio_timer_decrement_source
Define type of timer decrement.

Values:

enumerator kFLEXIO__TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO__TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO__TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO__TimerDecSrcOnTriggerInputShiftTriggerInput
Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum _flexio_timer_reset_condition
Define type of timer reset condition.

Values:

enumerator kFLEXIO__TimerResetNever

Timer never reset.

enumerator kFLEXIO__TimerResetOnTimerPinEqualToTimerOutput

Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerTriggerEqualToTimerOutput

Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO__TimerResetOnTimerPinRisingEdge

Timer reset on Timer Pin rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerRisingEdge

Timer reset on Trigger rising edge.

enumerator kFLEXIO__TimerResetOnTimerTriggerBothEdge

Timer reset on Trigger rising or falling edge.

enum _flexio_timer_disable_condition

Define type of timer disable condition.

Values:

enumerator kFLEXIO__TimerDisableNever

Timer never disabled.

enumerator kFLEXIO__TimerDisableOnPreTimerDisable

Timer disabled on Timer N-1 disable.

enumerator kFLEXIO__TimerDisableOnTimerCompare

Timer disabled on Timer compare.

enumerator kFLEXIO__TimerDisableOnTimerCompareTriggerLow

Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO__TimerDisableOnPinBothEdge

Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO__TimerDisableOnPinBothEdgeTriggerHigh

Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO__TimerDisableOnTriggerFallingEdge

Timer disabled on Trigger falling edge.

enum _flexio_timer_enable_condition

Define type of timer enable condition.

Values:

enumerator kFLEXIO__TimerEnabledAlways

Timer always enabled.

enumerator kFLEXIO__TimerEnableOnPrevTimerEnable

Timer enabled on Timer N-1 enable.

enumerator kFLEXIO__TimerEnableOnTriggerHigh

Timer enabled on Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerHighPinHigh

Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO__TimerEnableOnPinRisingEdge

Timer enabled on Pin rising edge.

enumerator kFLEXIO__TimerEnableOnPinRisingEdgeTriggerHigh
Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO__TimerEnableOnTriggerRisingEdge
Timer enabled on Trigger rising edge.

enumerator kFLEXIO__TimerEnableOnTriggerBothEdge
Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition
Define type of timer stop bit generate condition.

Values:

enumerator kFLEXIO__TimerStopBitDisabled
Stop bit disabled.

enumerator kFLEXIO__TimerStopBitEnableOnTimerCompare
Stop bit is enabled on timer compare.

enumerator kFLEXIO__TimerStopBitEnableOnTimerDisable
Stop bit is enabled on timer disable.

enumerator kFLEXIO__TimerStopBitEnableOnTimerCompareDisable
Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition
Define type of timer start bit generate condition.

Values:

enumerator kFLEXIO__TimerStartBitDisabled
Start bit disabled.

enumerator kFLEXIO__TimerStartBitEnabled
Start bit enabled.

enum _flexio_timer_output_state
FlexIO as PWM channel output state.

Values:

enumerator kFLEXIO__PwmLow
The output state of PWM channel is low

enumerator kFLEXIO__PwmHigh
The output state of PWM channel is high

enum _flexio_shifter_timer_polarity
Define type of timer polarity for shifter control.

Values:

enumerator kFLEXIO__ShifterTimerPolarityOnPositive
Shift on positive edge of shift clock.

enumerator kFLEXIO__ShifterTimerPolarityOnNegative
Shift on negative edge of shift clock.

enum _flexio_shifter_mode
Define type of shifter working mode.

Values:

enumerator kFLEXIO__ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO__ShifterModeReceive

Receive mode.

enumerator kFLEXIO__ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO__ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO__ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO__ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO__ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio_shifter_input_source

Define type of shifter input source.

Values:

enumerator kFLEXIO__ShifterInputFromPin

Shifter input from pin.

enumerator kFLEXIO__ShifterInputFromNextShifterOutput

Shifter input from Shifter N+1.

enum _flexio_shifter_stop_bit

Define of STOP bit configuration.

Values:

enumerator kFLEXIO__ShifterStopBitDisable

Disable shifter stop bit.

enumerator kFLEXIO__ShifterStopBitLow

Set shifter stop bit to logic low level.

enumerator kFLEXIO__ShifterStopBitHigh

Set shifter stop bit to logic high level.

enum _flexio_shifter_start_bit

Define type of START bit configuration.

Values:

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnEnable

Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO__ShifterStartBitDisabledLoadDataOnShift

Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO__ShifterStartBitLow

Set shifter start bit to logic low level.

enumerator kFLEXIO__ShifterStartBitHigh

Set shifter start bit to logic high level.

enum *_flexio_shifter_buffer_type*

Define FlexIO shifter buffer type.

Values:

enumerator *kFLEXIO__ShifterBuffer*

Shifter Buffer N Register.

enumerator *kFLEXIO__ShifterBufferBitSwapped*

Shifter Buffer N Bit Byte Swapped Register.

enumerator *kFLEXIO__ShifterBufferByteSwapped*

Shifter Buffer N Byte Swapped Register.

enumerator *kFLEXIO__ShifterBufferBitByteSwapped*

Shifter Buffer N Bit Swapped Register.

enumerator *kFLEXIO__ShifterBufferNibbleByteSwapped*

Shifter Buffer N Nibble Byte Swapped Register.

enumerator *kFLEXIO__ShifterBufferHalfWordSwapped*

Shifter Buffer N Half Word Swapped Register.

enumerator *kFLEXIO__ShifterBufferNibbleSwapped*

Shifter Buffer N Nibble Swapped Register.

enum *_flexio_gpio_direction*

FLEXIO gpio direction definition.

Values:

enumerator *kFLEXIO__DigitalInput*

Set current pin as digital input

enumerator *kFLEXIO__DigitalOutput*

Set current pin as digital output

enum *_flexio_pin_input_config*

FLEXIO gpio input config.

Values:

enumerator *kFLEXIO__InputInterruptDisabled*

Interrupt request is disabled.

enumerator *kFLEXIO__InputInterruptEnable*

Interrupt request is enable.

enumerator *kFLEXIO__FlagRisingEdgeEnable*

Input pin flag on rising edge.

enumerator *kFLEXIO__FlagFallingEdgeEnable*

Input pin flag on falling edge.

typedef enum *_flexio_timer_trigger_polarity* *flexio_timer_trigger_polarity_t*

Define time of timer trigger polarity.

typedef enum *_flexio_timer_trigger_source* *flexio_timer_trigger_source_t*

Define type of timer trigger source.

typedef enum *_flexio_pin_config* *flexio_pin_config_t*

Define type of timer/shifter pin configuration.

typedef enum *_flexio_pin_polarity* flexio_pin_polarity_t

Definition of pin polarity.

typedef enum *_flexio_timer_mode* flexio_timer_mode_t

Define type of timer work mode.

typedef enum *_flexio_timer_output* flexio_timer_output_t

Define type of timer initial output or timer reset condition.

typedef enum *_flexio_timer_decrement_source* flexio_timer_decrement_source_t

Define type of timer decrement.

typedef enum *_flexio_timer_reset_condition* flexio_timer_reset_condition_t

Define type of timer reset condition.

typedef enum *_flexio_timer_disable_condition* flexio_timer_disable_condition_t

Define type of timer disable condition.

typedef enum *_flexio_timer_enable_condition* flexio_timer_enable_condition_t

Define type of timer enable condition.

typedef enum *_flexio_timer_stop_bit_condition* flexio_timer_stop_bit_condition_t

Define type of timer stop bit generate condition.

typedef enum *_flexio_timer_start_bit_condition* flexio_timer_start_bit_condition_t

Define type of timer start bit generate condition.

typedef enum *_flexio_timer_output_state* flexio_timer_output_state_t

FlexIO as PWM channel output state.

typedef enum *_flexio_shifter_timer_polarity* flexio_shifter_timer_polarity_t

Define type of timer polarity for shifter control.

typedef enum *_flexio_shifter_mode* flexio_shifter_mode_t

Define type of shifter working mode.

typedef enum *_flexio_shifter_input_source* flexio_shifter_input_source_t

Define type of shifter input source.

typedef enum *_flexio_shifter_stop_bit* flexio_shifter_stop_bit_t

Define of STOP bit configuration.

typedef enum *_flexio_shifter_start_bit* flexio_shifter_start_bit_t

Define type of START bit configuration.

typedef enum *_flexio_shifter_buffer_type* flexio_shifter_buffer_type_t

Define FlexIO shifter buffer type.

typedef struct *_flexio_config* flexio_config_t

Define FlexIO user configuration structure.

typedef struct *_flexio_timer_config* flexio_timer_config_t

Define FlexIO timer configuration structure.

typedef struct *_flexio_shifter_config* flexio_shifter_config_t

Define FlexIO shifter configuration structure.

typedef enum *_flexio_gpio_direction* flexio_gpio_direction_t

FLEXIO gpio direction definition.

typedef enum *_flexio_pin_input_config* flexio_pin_input_config_t

FLEXIO gpio input config.

```
typedef struct flexio_gpio_config flexio_gpio_config_t
```

The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

```
typedef void (*flexio_isr_t)(void *base, void *handle)
```

typedef for FlexIO simulated driver interrupt handler.

```
FLEXIO_Type *const s_flexioBases[]
```

Pointers to flexio bases for each instance.

```
const clock_ip_name_t s_flexioClocks[]
```

Pointers to flexio clocks for each instance.

```
void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, flexio_gpio_config_t *config)
```

Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the FLEXIO_SetPinConfig() function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalInput,
    0U,
    kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
    kFLEXIO_DigitalOutput,
    0U,
    0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

```
FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)
```

Calculate FlexIO timer trigger.

```
FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(x)
```

```
FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)
```

```
struct flexio_config
```

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members

```
bool enableFlexio
```

Enable/disable FlexIO module

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

struct `_flexio_timer_config`

`#include <fsl_flexio.h>` Define FlexIO timer configuration structure.

Public Members

uint32_t triggerSelect

The internal trigger selection number using MACROs.

flexio_timer_trigger_polarity_t triggerPolarity

Trigger Polarity.

flexio_timer_trigger_source_t triggerSource

Trigger Source, internal (see 'trgsel') or external.

flexio_pin_config_t pinConfig

Timer Pin Configuration.

uint32_t pinSelect

Timer Pin number Select.

flexio_pin_polarity_t pinPolarity

Timer Pin Polarity.

flexio_timer_mode_t timerMode

Timer work Mode.

flexio_timer_output_t timerOutput

Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

flexio_timer_decrement_source_t timerDecrement

Configures the source of the Timer decrement and the source of the Shift clock.

flexio_timer_reset_condition_t timerReset

Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

flexio_timer_disable_condition_t timerDisable

Configures the condition that causes the Timer to be disabled and stop decrementing.

flexio_timer_enable_condition_t timerEnable

Configures the condition that causes the Timer to be enabled and start decrementing.

flexio_timer_stop_bit_condition_t timerStop

Timer STOP Bit generation.

flexio_timer_start_bit_condition_t timerStart

Timer STRAT Bit generation.

uint32_t timerCompare

Value for Timer Compare N Register.

struct `_flexio_shifter_config`

#include <fsl_flexio.h> Define FlexIO shifter configuration structure.

Public Members

uint32_t timerSelect

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

flexio_shifter_timer_polarity_t timerPolarity

Timer Polarity.

flexio_pin_config_t pinConfig

Shifter Pin Configuration.

uint32_t pinSelect

Shifter Pin number Select.

flexio_pin_polarity_t pinPolarity

Shifter Pin Polarity.

flexio_shifter_mode_t shifterMode

Configures the mode of the Shifter.

uint32_t parallelWidth

Configures the parallel width when using parallel mode.

flexio_shifter_input_source_t inputSource

Selects the input source for the shifter.

flexio_shifter_stop_bit_t shifterStop

Shifter STOP bit.

flexio_shifter_start_bit_t shifterStart

Shifter START bit.

struct `_flexio_gpio_config`

#include <fsl_flexio.h> The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

flexio_gpio_direction_t pinDirection

FLEXIO pin direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

uint8_t inputConfig

Set an input config

2.18 FlexIO eDMA I2S Driver

```
void FLEXIO_I2S_TransferTxCreateHandleEDMA(FLEXIO_I2S_Type *base,  
                                           flexio_i2s_edma_handle_t *handle,  
                                           flexio_i2s_edma_callback_t callback, void  
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferRxCreateHandleEDMA(FLEXIO_I2S_Type *base,  
                                           flexio_i2s_edma_handle_t *handle,  
                                           flexio_i2s_edma_callback_t callback, void  
                                           *userData, edma_handle_t *dmaHandle)
```

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

```
void FLEXIO_I2S_TransferSetFormatEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t  
                                       *handle, flexio_i2s_format_t *format, uint32_t  
                                       srcClock_Hz)
```

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

```
status_t FLEXIO_I2S_TransferSendEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                     *handle, flexio_i2s_transfer_t *xfer)
```

Performs a non-blocking FlexIO I2S transfer using DMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA send successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_TxBusy – FlexIO I2S is busy sending data.

```
status_t FLEXIO_I2S_TransferReceiveEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                         *handle, flexio_i2s_transfer_t *xfer)
```

Performs a non-blocking FlexIO I2S receive using eDMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA receive successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_RxBusy – FlexIO I2S is busy receiving data.

```
void FLEXIO_I2S_TransferAbortSendEDMA(FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t
                                       *handle)
```

Aborts a FlexIO I2S transfer using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

```
void FLEXIO_I2S_TransferAbortReceiveEDMA(FLEXIO_I2S_Type *base,
                                          flexio_i2s_edma_handle_t *handle)
```

Aborts a FlexIO I2S receive using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.

- handle – FlexIO I2S DMA handle pointer.

status_t FLEXIO_I2S_TransferGetSendCountEDMA(*FLEXIO_I2S_Type* *base,
flexio_i2s_edma_handle_t *handle, size_t
*count)

Gets the remaining bytes to be sent.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t FLEXIO_I2S_TransferGetReceiveCountEDMA(*FLEXIO_I2S_Type* *base,
flexio_i2s_edma_handle_t *handle, size_t
*count)

Get the remaining bytes to be received.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION

FlexIO I2S EDMA driver version 2.1.9.

typedef struct flexio_i2s_edma_handle flexio_i2s_edma_handle_t

typedef void (*flexio_i2s_edma_callback_t)(*FLEXIO_I2S_Type* *base, flexio_i2s_edma_handle_t
*handle, *status_t* status, void *userData)

FlexIO I2S eDMA transfer callback function for finish and error.

struct flexio_i2s_edma_handle

#include <fsl_flexio_i2s_edma.h> FlexIO I2S DMA transfer handle, users should not touch the
content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for FlexIO I2S send

uint8_t bytesPerFrame

Bytes in a frame

uint8_t nbytes

eDMA minor byte transfer count initially configured.

`uint32_t state`
Internal state for FlexIO I2S eDMA transfer

`flexio_i2s_edma_callback_t callback`
Callback for users while transfer finish or error occurred

`void *userData`
User callback parameter

`edma_tcd_t tcd[(4U) + 1U]`
TCD pool for eDMA transfer.

`flexio_i2s_transfer_t queue[(4U)]`
Transfer queue storing queued transfer.

`size_t transferSize[(4U)]`
Data bytes need to transfer

`volatile uint8_t queueUser`
Index for user to queue transfer.

`volatile uint8_t queueDriver`
Index for driver to get the transfer data and size

2.19 FlexIO eDMA MCU Interface LCD Driver

`status_t FLEXIO_MCULCD_TransferCreateHandleEDMA(FLEXIO_MCULCD_Type *base,
flexio_mculcd_edma_handle_t *handle,
flexio_mculcd_edma_transfer_callback_t
callback, void *userData,
edma_handle_t *txDmaHandle,
edma_handle_t *rxDmaHandle)`

Initializes the FLEXIO MCULCD master eDMA handle.

This function initializes the FLEXIO MCULCD master eDMA handle which can be used for other FLEXIO MCULCD transactional APIs. For a specified FLEXIO MCULCD instance, call this API once to get the initialized handle.

Parameters

- `base` – Pointer to `FLEXIO_MCULCD_Type` structure.
- `handle` – Pointer to `flexio_mculcd_edma_handle_t` structure to store the transfer state.
- `callback` – MCULCD transfer complete callback, NULL means no callback.
- `userData` – callback function parameter.
- `txDmaHandle` – User requested eDMA handle for FlexIO MCULCD eDMA TX, the DMA request source of this handle should be the first of TX shifters.
- `rxDmaHandle` – User requested eDMA handle for FlexIO MCULCD eDMA RX, the DMA request source of this handle should be the last of RX shifters.

Return values

`kStatus_Success` – Successfully create the handle.

`status_t FLEXIO_MCULCD_TransferEDMA(FLEXIO_MCULCD_Type *base,
flexio_mculcd_edma_handle_t *handle,
flexio_mculcd_transfer_t *xfer)`

Performs a non-blocking FlexIO MCULCD transfer using eDMA.

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

- a. Use the transfer completed callback;
- b. Polling function FLEXIO_MCULCD_GetTransferCountEDMA

Parameters

- base – pointer to FLEXIO_MCULCD_Type structure.
- handle – pointer to flexio_mculcd_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO MCULCD transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_MCULCD_Busy – FlexIO MCULCD is not idle, it is running another transfer.

```
void FLEXIO_MCULCD_TransferAbortEDMA(FLEXIO_MCULCD_Type *base,
                                     flexio_mculcd_edma_handle_t *handle)
```

Aborts a FlexIO MCULCD transfer using eDMA.

Parameters

- base – pointer to FLEXIO_MCULCD_Type structure.
- handle – FlexIO MCULCD eDMA handle pointer.

```
status_t FLEXIO_MCULCD_TransferGetCountEDMA(FLEXIO_MCULCD_Type *base,
                                             flexio_mculcd_edma_handle_t *handle,
                                             size_t *count)
```

Gets the remaining bytes for FlexIO MCULCD eDMA transfer.

Parameters

- base – pointer to FLEXIO_MCULCD_Type structure.
- handle – FlexIO MCULCD eDMA handle pointer.
- count – Number of count transferred so far by the eDMA transaction.

Return values

- kStatus_Success – Get the transferred count Successfully.
- kStatus_NoTransferInProgress – No transfer in process.

```
typedef struct flexio_mculcd_edma_handle flexio_mculcd_edma_handle_t
typedef for flexio_mculcd_edma_handle_t in advance.
```

```
typedef void (*flexio_mculcd_edma_transfer_callback_t)(FLEXIO_MCULCD_Type *base,
flexio_mculcd_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO MCULCD master callback for transfer complete.

When transfer finished, the callback function is called and returns the status as kStatus_FLEXIO_MCULCD_Idle.

```
FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION
```

FlexIO MCULCD EDMA driver version.

```
struct _flexio_mculcd_edma_handle
```

#include <fsl_flexio_mculcd_edma.h> FlexIO MCULCD eDMA transfer handle, users should not touch the content of the handle.

Public Members

FLEXIO_MCULCD_Type *base

Pointer to the FLEXIO_MCULCD_Type.

uint8_t txShifterNum

Number of shifters used for TX.

uint8_t rxShifterNum

Number of shifters used for RX.

uint32_t minorLoopBytes

eDMA transfer minor loop bytes.

edma_modulo_t txEdmaModulo

Modulo value for the FlexIO shifter buffer access.

edma_modulo_t rxEdmaModulo

Modulo value for the FlexIO shifter buffer access.

uint32_t dataAddrOrSameValue

When sending the same value for many times, this is the value to send. When writing or reading array, this is the address of the data array.

size_t dataCount

Total count to be transferred.

volatile size_t remainingCount

Remaining count still not transfered.

volatile uint32_t state

FlexIO MCULCD driver internal state.

edma_handle_t *txDmaHandle

DMA handle for MCULCD TX

edma_handle_t *rxDmaHandle

DMA handle for MCULCD RX

flexio_mculcd_edma_transfer_callback_t completionCallback

Callback for MCULCD DMA transfer

void *userData

User Data for MCULCD DMA callback

2.20 FlexIO eDMA SPI Driver

```
status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                    flexio_spi_master_edma_handle_t  
                                                    *handle,  
                                                    flexio_spi_master_edma_transfer_callback_t  
                                                    callback, void *userData,  
                                                    edma_handle_t *txHandle,  
                                                    edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI master eDMA handle.

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

```
status_t FLEXIO_SPI_MasterTransferEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle,  
                                         flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

```
void FLEXIO_SPI_MasterTransferAbortEDMA(FLEXIO_SPI_Type *base,  
                                         flexio_spi_master_edma_handle_t *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

```
status_t FLEXIO_SPI_MasterTransferGetCountEDMA(FLEXIO_SPI_Type *base,  
                                              flexio_spi_master_edma_handle_t *handle,  
                                              size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

```
static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(FLEXIO_SPI_Type *base,  
                                                         flexio_spi_slave_edma_handle_t  
                                                         *handle,  
                                                         flexio_spi_slave_edma_transfer_callback_t  
                                                         callback, void *userData,  
                                                         edma_handle_t *txHandle,  
                                                         edma_handle_t *rxHandle)
```

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

```
status_t FLEXIO_SPI_SlaveTransferEDMA(FLEXIO_SPI_Type *base,  
                                      flexio_spi_slave_edma_handle_t *handle,  
                                      flexio_spi_transfer_t *xfer)
```

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(FLEXIO_SPI_Type *base,
                                                    flexio_spi_slave_edma_handle_t
                                                    *handle)
```

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to *FLEXIO_SPI_Type* structure.
- handle – Pointer to *flexio_spi_slave_edma_handle_t* structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCountEDMA(FLEXIO_SPI_Type *base,
                                                            flexio_spi_slave_edma_handle_t
                                                            *handle, size_t *count)
```

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- base – Pointer to *FLEXIO_SPI_Type* structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

```
FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION
```

FlexIO SPI EDMA driver version.

```
typedef struct flexio_spi_master_edma_handle flexio_spi_master_edma_handle_t
typedef for flexio_spi_master_edma_handle_t in advance.
```

```
typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t
Slave handle is the same with master handle.
```

```
typedef void (*flexio_spi_master_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_master_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI master callback for finished transmit.

```
typedef void (*flexio_spi_slave_edma_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_slave_edma_handle_t *handle, status_t status, void *userData)
```

FlexIO SPI slave callback for finished transmit.

```
struct flexio_spi_master_edma_handle
#include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch
the content of the handle.
```

Public Members

```
size_t transferSize
```

Total bytes to be transferred.

```
uint8_t nbytes
```

eDMA minor byte transfer count initially configured.

```
bool txInProgress
```

Send transfer in progress

```
bool rxInProgress
```

Receive transfer in progress

```
edma_handle_t *txHandle
```

DMA handler for SPI send

edma_handle_t *rxHandle
DMA handler for SPI receive

flexio_spi_master_edma_transfer_callback_t callback
Callback for SPI DMA transfer

void *userData
User Data for SPI DMA callback

2.21 FlexIO eDMA UART Driver

status_t FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base,
 flexio_uart_edma_handle_t *handle,
 flexio_uart_edma_transfer_callback_t
 callback, void *userData, *edma_handle_t*
 *txEdmaHandle, *edma_handle_t*
 *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_UART_Type.
- handle – Pointer to flexio_uart_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base,
 flexio_uart_edma_handle_t *handle,
 flexio_uart_transfer_t *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

status_t FLEXIO_UART_TransferReceiveEDMA(*FLEXIO_UART_Type* *base,
 flexio_uart_edma_handle_t *handle,
 flexio_uart_transfer_t *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see flexio_uart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

```
void FLEXIO_UART_TransferAbortSendEDMA(FLEXIO_UART_Type *base,  
                                         flexio_uart_edma_handle_t *handle)
```

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
void FLEXIO_UART_TransferAbortReceiveEDMA(FLEXIO_UART_Type *base,  
                                           flexio_uart_edma_handle_t *handle)
```

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

```
status_t FLEXIO_UART_TransferGetSendCountEDMA(FLEXIO_UART_Type *base,  
                                                flexio_uart_edma_handle_t *handle,  
                                                size_t *count)
```

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

```
status_t FLEXIO_UART_TransferGetReceiveCountEDMA(FLEXIO_UART_Type *base,  
                                                  flexio_uart_edma_handle_t *handle,  
                                                  size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- `base` – Pointer to `FLEXIO_UART_Type`
- `handle` – Pointer to `flexio_uart_edma_handle_t` structure
- `count` – Number of bytes received so far by the non-blocking transaction.

Return values

- `kStatus_NoTransferInProgress` – transfer has finished or no transfer in progress.
- `kStatus_Success` – Successfully return the count.

`FSL_FLEXIO_UART_EDMA_DRIVER_VERSION`

FlexIO UART EDMA driver version.

`typedef struct flexio_uart_edma_handle flexio_uart_edma_handle_t`

`typedef void (*flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base,
flexio_uart_edma_handle_t *handle, status_t status, void *userData)`

UART transfer callback function.

`struct _flexio_uart_edma_handle`

`#include <fsl_flexio_uart_edma.h>` UART eDMA handle.

Public Members

`flexio_uart_edma_transfer_callback_t` callback

Callback function.

`void *userData`

UART callback function parameter.

`size_t txDataSizeAll`

Total bytes to be sent.

`size_t rxDataSizeAll`

Total bytes to be received.

`edma_handle_t *txEdmaHandle`

The eDMA TX channel used.

`edma_handle_t *rxEdmaHandle`

The eDMA RX channel used.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

`volatile uint8_t txState`

TX transfer state.

`volatile uint8_t rxState`

RX transfer state

2.22 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, *uint32_t* srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifter and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;  
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)
Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)
Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND _flexio_i2c_master_status_flags can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)
Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_I2C_RxFullFlag
 - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)
Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
 - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)
Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps,
uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ

- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t* direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
 - kFLEXIO_I2C_Write: Transmit
 - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

- `count` – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- `kStatus_Success` – Successfully configured the count.
- `kStatus_InvalidArgument` – Input argument is invalid.

`static inline void FLEXIO_I2C_MasterWriteByte(FLEXIO_I2C_Type *base, uint32_t data)`

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the `TxEEmptyFlag` is asserted before calling this API.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.
- `data` – a byte of data.

`static inline uint8_t FLEXIO_I2C_MasterReadByte(FLEXIO_I2C_Type *base)`

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.

Returns

data byte read.

`status_t FLEXIO_I2C_MasterWriteBlocking(FLEXIO_I2C_Type *base, const uint8_t *txBuff, uint8_t txSize)`

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.
- `txBuff` – The data bytes to send.
- `txSize` – The number of data bytes to send.

Return values

- `kStatus_Success` – Successfully write data.
- `kStatus_FLEXIO_I2C_Nak` – Receive NAK during writing data.
- `kStatus_FLEXIO_I2C_Timeout` – Timeout polling status flags.

`status_t FLEXIO_I2C_MasterReadBlocking(FLEXIO_I2C_Type *base, uint8_t *rxBuff, uint8_t rxSize)`

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus__Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_transfer_t *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_handle_t *handle,
flexio_i2c_master_transfer_callback_t
callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus__Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base,
flexio_i2c_master_handle_t *handle,
flexio_i2c_master_transfer_t *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the

transfer is finished. If the return status is not `kStatus_FLEXIO_I2C_Busy`, the transfer is finished.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state
- `xfer` – pointer to `flexio_i2c_master_transfer_t` structure

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_FLEXIO_I2C_Busy` – FlexIO I2C is not idle, is running another transfer.

`status_t FLEXIO_I2C_MasterTransferGetCount(FLEXIO_I2C_Type *base,
flexio_i2c_master_handle_t *handle, size_t
*count)`

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure.
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – `count` is Invalid.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_I2C_MasterTransferAbort(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t
*handle)`

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – Pointer to `FLEXIO_I2C_Type` structure
- `handle` – Pointer to `flexio_i2c_master_handle_t` structure which stores the transfer state

`void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)`

Master interrupt handler.

Parameters

- `i2cType` – Pointer to `FLEXIO_I2C_Type` structure
- `i2cHandle` – Pointer to `flexio_i2c_master_transfer_t` structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak

NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout

Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable

Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable

Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag

Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag

Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag

Receive NAK flag.

enum _flexio_i2c_direction

Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write

Master send to slave.

enumerator kFLEXIO_I2C_Read

Master receive from slave.

typedef enum _flexio_i2c_direction flexio_i2c_direction_t

Direction of master transfer.

typedef struct _flexio_i2c_type FLEXIO_I2C_Type

Define FlexIO I2C master access structure typedef.

typedef struct _flexio_i2c_master_config flexio_i2c_master_config_t

Define FlexIO I2C master user configuration structure.

```
typedef struct _flexio_i2c_master_transfer flexio_i2c_master_transfer_t
```

Define FlexIO I2C master transfer structure.

```
typedef struct _flexio_i2c_master_handle flexio_i2c_master_handle_t
```

FlexIO I2C master handle typedef.

```
typedef void (*flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base,  
flexio_i2c_master_handle_t *handle, status_t status, void *userData)
```

FlexIO I2C master transfer callback typedef.

```
I2C_RETRY_TIMES
```

Retry times for waiting flag.

```
struct _flexio_i2c_type
```

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

uint8_t SDAPinIndex

Pin select for I2C SDA.

uint8_t SCLPinIndex

Pin select for I2C SCL.

uint8_t shifterIndex[2]

Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]

Timer index used in FlexIO I2C.

uint32_t baudrate

Master transfer baudrate, used to calculate delay time.

```
struct _flexio_i2c_master_config
```

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster

Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in Bps.

```
struct _flexio_i2c_master_transfer
```

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags

Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress

7-bit slave address.

flexio_i2c_direction_t direction

Transfer direction, read or write.

uint32_t subaddress

Sub address. Transferred MSB first.

uint8_t subaddressSize

Size of sub address.

uint8_t volatile *data

Transfer buffer.

volatile size_t dataSize

Transfer size.

struct *_flexio_i2c_master_handle*

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer

FlexIO I2C master transfer copy.

size_t transferSize

Total bytes to be transferred.

uint8_t state

Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback

Callback function called at transfer event. Callback function called at transfer event.

void *userData

Callback parameter passed to callback function.

bool needRestart

Whether master needs to send re-start signal.

2.23 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by FLEXIO_I2S_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in FLEXIO_I2S_Init(). Users may use the initialized structure unchanged in FLEXIO_I2S_Init() or modify some fields of the structure before calling FLEXIO_I2S_Init().

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXIO_I2S_Init to use the FlexIO I2S module.

Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO_I2S_Enable(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S module operation.

Parameters

- base – Pointer to FLEXIO_I2S_Type
- enable – True to enable, false dose not have any effect.

uint32_t FLEXIO_I2S_GetStatusFlags(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S status flags.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

Status flag, which are ORed by the enumerators in the _flexio_i2s_status_flags.

void FLEXIO_I2S_EnableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

void FLEXIO_I2S_DisableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

static inline void FLEXIO_I2S_TxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)
Enables/disables the FlexIO I2S Tx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO_I2S_RxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)
Enables/disables the FlexIO I2S Rx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)
Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s send data register address.

static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)
Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure

Returns

FlexIO i2s receive data register address.

void FLEXIO_I2S_MasterSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format,
uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – I2S master clock source frequency in Hz.

void FLEXIO_I2S_SlaveSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format)
Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- format – Pointer to FlexIO I2S audio data format structure.

```
status_t FLEXIO_I2S_WriteBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *txData,  
                                size_t size)
```

Sends data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

```
static inline void FLEXIO_I2S_WriteData(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint32_t  
                                       data)
```

Writes data into a data register.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- data – Data to be written.

```
status_t FLEXIO_I2S_ReadBlocking(FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *rxData,  
                                size_t size)
```

Receives a piece of data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- rxData – Pointer to the data to be read.
- size – Bytes to be read.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

```
static inline uint32_t FLEXIO_I2S_ReadData(FLEXIO_I2S_Type *base)
```

Reads a data from the data register.

Parameters

- base – FlexIO I2S base pointer

Returns

Data read from data register.

```
void FLEXIO_I2S_TransferTxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
                                       flexio_i2s_callback_t callback, void *userData)
```

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure
- handle – Pointer to *flexio_i2s_handle_t* structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

```
void FLEXIO_I2S_TransferSetFormat(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
                                  flexio_i2s_format_t *format, uint32_t srcClock_Hz)
```

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

```
void FLEXIO_I2S_TransferRxCreateHandle(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
                                       flexio_i2s_callback_t callback, void *userData)
```

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.
- handle – Pointer to *flexio_i2s_handle_t* structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

```
status_t FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t  
                                             *handle, flexio_i2s_transfer_t *xfer)
```

Performs an interrupt non-blocking send transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call *FLEXIO_I2S_GetRemainingBytes* to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to *FLEXIO_I2S_Type* structure.

- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state
- `xfer` – Pointer to `flexio_i2s_transfer_t` structure

Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_FLEXIO_I2S_TxBusy` – Previous transmission still not finished, data not all written to TX register yet.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`status_t FLEXIO_I2S_TransferReceiveNonBlocking(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_transfer_t *xfer)`

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call `FLEXIO_I2S_GetRemainingBytes` to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state
- `xfer` – Pointer to `flexio_i2s_transfer_t` structure

Return values

- `kStatus_Success` – Successfully start the data receive.
- `kStatus_FLEXIO_I2S_RxBusy` – Previous receive still not finished.
- `kStatus_InvalidArgument` – The input parameter is invalid.

`void FLEXIO_I2S_TransferAbortSend(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)`
Aborts the current send.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state

`void FLEXIO_I2S_TransferAbortReceive(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)`
Aborts the current receive.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.

- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state

`status_t FLEXIO_I2S_TransferGetSendCount(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)`

Gets the remaining bytes to be sent.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state
- `count` – Bytes sent.

Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`status_t FLEXIO_I2S_TransferGetReceiveCount(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)`

Gets the remaining bytes to be received.

Parameters

- `base` – Pointer to `FLEXIO_I2S_Type` structure.
- `handle` – Pointer to `flexio_i2s_handle_t` structure which stores the transfer state
- `count` – Bytes recieved.

Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

Returns

`count` Bytes received.

`void FLEXIO_I2S_TransferTxHandleIRQ(void *i2sBase, void *i2sHandle)`

Tx interrupt handler.

Parameters

- `i2sBase` – Pointer to `FLEXIO_I2S_Type` structure.
- `i2sHandle` – Pointer to `flexio_i2s_handle_t` structure

`void FLEXIO_I2S_TransferRxHandleIRQ(void *i2sBase, void *i2sHandle)`

Rx interrupt handler.

Parameters

- `i2sBase` – Pointer to `FLEXIO_I2S_Type` structure.
- `i2sHandle` – Pointer to `flexio_i2s_handle_t` structure.

`FSL_FLEXIO_I2S_DRIVER_VERSION`

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator kStatus_FLEXIO_I2S_Idle

FlexIO I2S is in idle state

enumerator kStatus_FLEXIO_I2S_TxBusy

FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_RxBusy

FlexIO I2S Rx is busy

enumerator kStatus_FLEXIO_I2S_Error

FlexIO I2S error occurred

enumerator kStatus_FLEXIO_I2S_QueueFull

FlexIO I2S transfer queue is full.

enumerator kStatus_FLEXIO_I2S_Timeout

FlexIO I2S timeout polling status flags.

enum _flexio_i2s_master_slave

Master or slave mode.

Values:

enumerator kFLEXIO_I2S_Master

Master mode

enumerator kFLEXIO_I2S_Slave

Slave mode

_flexio_i2s_interrupt_enable Define FlexIO FlexIO I2S interrupt mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_I2S_RxDataRegFullInterruptEnable

Receive buffer full interrupt enable.

_flexio_i2s_status_flags Define FlexIO FlexIO I2S status mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO_I2S_RxDataRegFullFlag

Receive buffer full flag.

enum _flexio_i2s_sample_rate

Audio sample rate.

Values:

enumerator kFLEXIO_I2S_SampleRate8KHz

Sample rate 8000Hz

```
enumerator kFLEXIO_I2S_SampleRate11025Hz
    Sample rate 11025Hz
enumerator kFLEXIO_I2S_SampleRate12KHz
    Sample rate 12000Hz
enumerator kFLEXIO_I2S_SampleRate16KHz
    Sample rate 16000Hz
enumerator kFLEXIO_I2S_SampleRate22050Hz
    Sample rate 22050Hz
enumerator kFLEXIO_I2S_SampleRate24KHz
    Sample rate 24000Hz
enumerator kFLEXIO_I2S_SampleRate32KHz
    Sample rate 32000Hz
enumerator kFLEXIO_I2S_SampleRate44100Hz
    Sample rate 44100Hz
enumerator kFLEXIO_I2S_SampleRate48KHz
    Sample rate 48000Hz
enumerator kFLEXIO_I2S_SampleRate96KHz
    Sample rate 96000Hz
enum _flexio_i2s_word_width
    Audio word width.
    Values:
enumerator kFLEXIO_I2S_WordWidth8bits
    Audio data width 8 bits
enumerator kFLEXIO_I2S_WordWidth16bits
    Audio data width 16 bits
enumerator kFLEXIO_I2S_WordWidth24bits
    Audio data width 24 bits
enumerator kFLEXIO_I2S_WordWidth32bits
    Audio data width 32 bits
typedef struct _flexio_i2s_type FLEXIO_I2S_Type
    Define FlexIO I2S access structure typedef.
typedef enum _flexio_i2s_master_slave flexio_i2s_master_slave_t
    Master or slave mode.
typedef struct _flexio_i2s_config flexio_i2s_config_t
    FlexIO I2S configure structure.
typedef struct _flexio_i2s_format flexio_i2s_format_t
    FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.
typedef enum _flexio_i2s_sample_rate flexio_i2s_sample_rate_t
    Audio sample rate.
typedef enum _flexio_i2s_word_width flexio_i2s_word_width_t
    Audio word width.
```

```
typedef struct _flexio_i2s_transfer flexio_i2s_transfer_t
```

Define FlexIO I2S transfer structure.

```
typedef struct _flexio_i2s_handle flexio_i2s_handle_t
```

```
typedef void (*flexio_i2s_callback_t)(FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle,  
status_t status, void *userData)
```

FlexIO I2S xfer callback prototype.

```
I2S_RETRY_TIMES
```

Retry times for waiting flag.

```
FLEXIO_I2S_XFER_QUEUE_SIZE
```

FlexIO I2S transfer queue size, user can refine it according to use case.

```
struct _flexio_i2s_type
```

#include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer

uint8_t txPinIndex

Tx data pin index in FlexIO pins

uint8_t rxPinIndex

Rx data pin index

uint8_t bclkPinIndex

Bit clock pin index

uint8_t fsPinIndex

Frame sync pin index

uint8_t txShifterIndex

Tx data shifter index

uint8_t rxShifterIndex

Rx data shifter index

uint8_t bclkTimerIndex

Bit clock timer index

uint8_t fsTimerIndex

Frame sync timer index

```
struct _flexio_i2s_config
```

#include <fsl_flexio_i2s.h> FlexIO I2S configure structure.

Public Members

bool enableI2S

Enable FlexIO I2S

flexio_i2s_master_slave_t masterSlave

Master or slave

flexio_pin_polarity_t txPinPolarity

Tx data pin polarity, active high or low

flexio_pin_polarity_t rxPinPolarity
Rx data pin polarity

flexio_pin_polarity_t bclkPinPolarity
Bit clock pin polarity

flexio_pin_polarity_t fsPinPolarity
Frame sync pin polarity

flexio_shifter_timer_polarity_t txTimerPolarity
Tx data valid on bclk rising or falling edge

flexio_shifter_timer_polarity_t rxTimerPolarity
Rx data valid on bclk rising or falling edge

struct *_flexio_i2s_format*

#include <fsl_flexio_i2s.h> FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

Public Members

uint8_t bitWidth
Bit width of audio data, always 8/16/24/32 bits

uint32_t sampleRate_Hz
Sample rate of the audio data

struct *_flexio_i2s_transfer*

#include <fsl_flexio_i2s.h> Define FlexIO I2S transfer structure.

Public Members

uint8_t *data
Data buffer start pointer

size_t dataSize
Bytes to be transferred.

struct *_flexio_i2s_handle*

#include <fsl_flexio_i2s.h> Define FlexIO I2S handle structure.

Public Members

uint32_t state
Internal state

flexio_i2s_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32bits

flexio_i2s_transfer_t queue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

2.24 FlexIO MCU Interface LCD Driver

status_t FLEXIO_MCULCD_Init(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_config_t* *config, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with FlexIO MCULCD configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_MCULCD_GetDefaultConfig.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- config – Pointer to the flexio_mculcd_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization success.
- kStatus_InvalidArgument – Initialization failed because of invalid argument.

void FLEXIO_MCULCD_Deinit(*FLEXIO_MCULCD_Type* *base)

Resets the FLEXIO_MCULCD timer and shifter configuration.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_GetDefaultConfig(*flexio_mculcd_config_t* *config)

Gets the default configuration to configure the FlexIO MCULCD.

The default configuration value is:

```
config->enable = true;  
config->enableInDoze = false;  
config->enableInDebug = true;  
config->enableFastAccess = true;  
config->baudRate_Bps = 96000000U;
```

Parameters

- config – Pointer to the flexio_mculcd_config_t structure.

uint32_t FLEXIO_MCULCD_GetStatusFlags(*FLEXIO_MCULCD_Type* *base)

Gets FlexIO MCULCD status flags.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

status flag; OR'ed value or the _flexio_mculcd_status_flags.

void FLEXIO_MCULCD_ClearStatusFlags(*FLEXIO_MCULCD_Type* *base, uint32_t mask)
Clears FlexIO MCULCD status flags.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- mask – Status to clear, it is the OR'ed value of _flexio_mculcd_status_flags.

void FLEXIO_MCULCD_EnableInterrupts(*FLEXIO_MCULCD_Type* *base, uint32_t mask)
Enables the FlexIO MCULCD interrupt.
This function enables the FlexIO MCULCD interrupt.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- mask – Interrupts to enable, it is the OR'ed value of _flexio_mculcd_interrupt_enable.

void FLEXIO_MCULCD_DisableInterrupts(*FLEXIO_MCULCD_Type* *base, uint32_t mask)
Disables the FlexIO MCULCD interrupt.
This function disables the FlexIO MCULCD interrupt.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- mask – Interrupts to disable, it is the OR'ed value of _flexio_mculcd_interrupt_enable.

static inline void FLEXIO_MCULCD_EnableTxDMA(*FLEXIO_MCULCD_Type* *base, bool enable)
Enables/disables the FlexIO MCULCD transmit DMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO_MCULCD_EnableRxDMA(*FLEXIO_MCULCD_Type* *base, bool enable)
Enables/disables the FlexIO MCULCD receive DMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress(*FLEXIO_MCULCD_Type* *base)
Gets the FlexIO MCULCD transmit data register address.

Gets the FlexIO MCULCD transmit data register address.

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

FlexIO MCULCD transmit data register address.

```
static inline uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress(FLEXIO_MCULCD_Type  
                                                             *base)
```

Gets the FlexIO MCULCD receive data register address.

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

FlexIO MCULCD receive data register address.

```
status_t FLEXIO_MCULCD_SetBaudRate(FLEXIO_MCULCD_Type *base, uint32_t  
                                   baudRate_Bps, uint32_t srcClock_Hz)
```

Set desired baud rate.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- baudRate_Bps – Desired baud rate in bit-per-second for all data lines combined.
- srcClock_Hz – FLEXIO clock frequency in Hz.

Return values

- kStatus_Success – Set successfully.
- kStatus_InvalidArgument – Could not set the baud rate.

```
void FLEXIO_MCULCD_SetSingleBeatWriteConfig(FLEXIO_MCULCD_Type *base)
```

Configures the FLEXIO MCULCD to multiple beats write mode.

At the beginning multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by FLEXIO_MCULCD_ClearSingleBeatWriteConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

```
void FLEXIO_MCULCD_ClearSingleBeatWriteConfig(FLEXIO_MCULCD_Type *base)
```

Clear the FLEXIO MCULCD multiple beats write mode configuration.

Clear the write configuration set by FLEXIO_MCULCD_SetSingleBeatWriteConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_SetSingleBeatReadConfig(*FLEXIO_MCULCD_Type* *base)

Configures the FLEXIO MCULCD to multiple beats read mode.

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD_ClearSingleBeatReadConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_ClearSingleBeatReadConfig(*FLEXIO_MCULCD_Type* *base)

Clear the FLEXIO MCULCD multiple beats read mode configuration.

Clear the read configuration set by FLEXIO_MCULCD_SetSingleBeatReadConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_SetMultiBeatsWriteConfig(*FLEXIO_MCULCD_Type* *base)

Configures the FLEXIO MCULCD to multiple beats write mode.

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by FLEXIO_MCULCD_ClearMultBeatsWriteConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig(*FLEXIO_MCULCD_Type* *base)

Clear the FLEXIO MCULCD multiple beats write mode configuration.

Clear the write configuration set by FLEXIO_MCULCD_SetMultBeatsWriteConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_SetMultiBeatsReadConfig(*FLEXIO_MCULCD_Type* *base)

Configures the FLEXIO MCULCD to multiple beats read mode.

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD_ClearMultBeatsReadConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_ClearMultiBeatsReadConfig(*FLEXIO_MCULCD_Type* *base)

Clear the FLEXIO MCULCD multiple beats read mode configuration.

Clear the read configuration set by FLEXIO_MCULCD_SetMultBeatsReadConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

static inline void FLEXIO_MCULCD_Enable(*FLEXIO_MCULCD_Type* *base, bool enable)

Enables/disables the FlexIO MCULCD module operation.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.
- enable – True to enable, false does not have any effect.

uint32_t FLEXIO_MCULCD_ReadData(*FLEXIO_MCULCD_Type* *base)

Read data from the FLEXIO MCULCD RX shifter buffer.

Read data from the RX shift buffer directly, it does no check whether the buffer is empty or not.

If the data bus width is 8-bit:

```
uint8_t value;  
value = (uint8_t)FLEXIO_MCULCD_ReadData(base);
```

If the data bus width is 16-bit:

```
uint16_t value;  
value = (uint16_t)FLEXIO_MCULCD_ReadData(base);
```

Note: This function returns the RX shifter buffer value (32-bit) directly. The return value should be converted according to data bus width.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

The data read out.

static inline void FLEXIO_MCULCD_WriteData(*FLEXIO_MCULCD_Type* *base, uint32_t data)

Write data into the FLEXIO MCULCD TX shifter buffer.

Write data into the TX shift buffer directly, it does no check whether the buffer is full or not.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- data – The data to write.

```
static inline void FLEXIO_MCULCD_StartTransfer(FLEXIO_MCULCD_Type *base)
```

Assert the nCS to start transfer.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

```
static inline void FLEXIO_MCULCD_StopTransfer(FLEXIO_MCULCD_Type *base)
```

De-assert the nCS to stop transfer.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

```
void FLEXIO_MCULCD_WaitTransmitComplete(void)
```

Wait for transmit data send out finished.

Currently there is no effective method to wait for the data send out from the shiter, so here use a while loop to wait.

Note: This is an internal used function.

```
void FLEXIO_MCULCD_WriteCommandBlocking(FLEXIO_MCULCD_Type *base, uint32_t  
                                         command)
```

Send command in blocking way.

This function sends the command and returns when the command has been sent out.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- command – The command to send.

```
void FLEXIO_MCULCD_WriteDataArrayBlocking(FLEXIO_MCULCD_Type *base, const void  
                                           *data, size_t size)
```

Send data array in blocking way.

This function sends the data array and returns when the data sent out.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- data – The data array to send.
- size – How many bytes to write.

```
void FLEXIO_MCULCD_ReadDataArrayBlocking(FLEXIO_MCULCD_Type *base, void *data,  
                                          size_t size)
```

Read data into array in blocking way.

This function reads the data into array and returns when the data read finished.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- data – The array to save the data.
- size – How many bytes to read.

```
void FLEXIO_MCULCD_WriteSameValueBlocking(FLEXIO_MCULCD_Type *base, uint32_t
                                         sameValue, size_t size)
```

Send the same value many times in blocking way.

This function sends the same value many times. It could be used to clear the LCD screen. If the data bus width is 8, this function will send LSB 8 bits of `sameValue` for `size` times. If the data bus is 16, this function will send LSB 16 bits of `sameValue` for `size / 2` times.

Parameters

- `base` – Pointer to the `FLEXIO_MCULCD_Type` structure.
- `sameValue` – The same value to send.
- `size` – How many bytes to send.

```
void FLEXIO_MCULCD_TransferBlocking(FLEXIO_MCULCD_Type *base,
                                     flexio_mculcd_transfer_t *xfer)
```

Performs a polling transfer.

Note: The API does not return until the transfer finished.

Parameters

- `base` – pointer to `FLEXIO_MCULCD_Type` structure.
- `xfer` – pointer to `flexio_mculcd_transfer_t` structure.

```
status_t FLEXIO_MCULCD_TransferCreateHandle(FLEXIO_MCULCD_Type *base,
                                             flexio_mculcd_handle_t *handle,
                                             flexio_mculcd_transfer_callback_t callback,
                                             void *userData)
```

Initializes the FlexIO MCULCD handle, which is used in transactional functions.

Parameters

- `base` – Pointer to the `FLEXIO_MCULCD_Type` structure.
- `handle` – Pointer to the `flexio_mculcd_handle_t` structure to store the transfer state.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

```
status_t FLEXIO_MCULCD_TransferNonBlocking(FLEXIO_MCULCD_Type *base,
                                             flexio_mculcd_handle_t *handle,
                                             flexio_mculcd_transfer_t *xfer)
```

Transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- `base` – Pointer to the `FLEXIO_MCULCD_Type` structure.
- `handle` – Pointer to the `flexio_mculcd_handle_t` structure to store the transfer state.
- `xfer` – FlexIO MCULCD transfer structure. See `flexio_mculcd_transfer_t`.

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_MCULCD_Busy` – MCULCD is busy with another transfer.

```
void FLEXIO_MCULCD_TransferAbort(FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle)
```

Aborts the data transfer, which used IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_MCULCD_Type` structure.
- `handle` – Pointer to the `flexio_mculcd_handle_t` structure to store the transfer state.

```
status_t FLEXIO_MCULCD_TransferGetCount(FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, size_t *count)
```

Gets the data transfer status which used IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_MCULCD_Type` structure.
- `handle` – Pointer to the `flexio_mculcd_handle_t` structure to store the transfer state.
- `count` – How many bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_Success` – Get the transferred count Successfully.
- `kStatus_NoTransferInProgress` – No transfer in process.

```
void FLEXIO_MCULCD_TransferHandleIRQ(void *base, void *handle)
```

FlexIO MCULCD IRQ handler function.

Parameters

- `base` – Pointer to the `FLEXIO_MCULCD_Type` structure.
- `handle` – Pointer to the `flexio_mculcd_handle_t` structure to store the transfer state.

```
FSL_FLEXIO_MCULCD_DRIVER_VERSION
```

FlexIO MCULCD driver version.

FlexIO LCD transfer status.

Values:

enumerator `kStatus_FLEXIO_MCULCD_Idle`

FlexIO LCD is idle.

enumerator `kStatus_FLEXIO_MCULCD_Busy`

FlexIO LCD is busy

enumerator `kStatus_FLEXIO_MCULCD_Error`

FlexIO LCD error occurred

enum _flexio_mculcd_pixel_format

Define FlexIO MCULCD pixel format.

Values:

enumerator kFLEXIO_MCULCD_RGB565
RGB565, 16-bit.

enumerator kFLEXIO_MCULCD_BGR565
BGR565, 16-bit.

enumerator kFLEXIO_MCULCD_RGB888
RGB888, 24-bit.

enumerator kFLEXIO_MCULCD_BGR888
BGR888, 24-bit.

enum _flexio_mculcd_bus

Define FlexIO MCULCD bus type.

Values:

enumerator kFLEXIO_MCULCD_8080
Using Intel 8080 bus.

enumerator kFLEXIO_MCULCD_6800
Using Motorola 6800 bus.

enum _flexio_mculcd_interrupt_enable

Define FlexIO MCULCD interrupt mask.

Values:

enumerator kFLEXIO_MCULCD_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_MCULCD_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_mculcd_status_flags

Define FlexIO MCULCD status mask.

Values:

enumerator kFLEXIO_MCULCD_TxEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_MCULCD_RxFullFlag
Receive buffer full flag.

enum _flexio_mculcd_dma_enable

Define FlexIO MCULCD DMA mask.

Values:

enumerator kFLEXIO_MCULCD_TxDmaEnable
Tx DMA request source

enumerator kFLEXIO_MCULCD_RxDmaEnable
Rx DMA request source

enum _flexio_mculcd_transfer_mode

Transfer mode.

Values:

enumerator `kFLEXIO_MCULCD_ReadArray`
Read data into an array.

enumerator `kFLEXIO_MCULCD_WriteArray`
Write data from an array.

enumerator `kFLEXIO_MCULCD_WriteSameValue`
Write the same value many times.

typedef enum *flexio_mculcd_pixel_format* flexio_mculcd_pixel_format_t
Define FlexIO MCULCD pixel format.

typedef enum *flexio_mculcd_bus* flexio_mculcd_bus_t
Define FlexIO MCULCD bus type.

typedef void (*flexio_mculcd_pin_func_t)(bool set)
Function to set or clear the CS and RS pin.

typedef struct *flexio_mculcd_type* FLEXIO_MCULCD_Type
Define FlexIO MCULCD access structure typedef.

typedef struct *flexio_mculcd_config* flexio_mculcd_config_t
Define FlexIO MCULCD configuration structure.

typedef enum *flexio_mculcd_transfer_mode* flexio_mculcd_transfer_mode_t
Transfer mode.

typedef struct *flexio_mculcd_transfer* flexio_mculcd_transfer_t
Define FlexIO MCULCD transfer structure.

typedef struct *flexio_mculcd_handle* flexio_mculcd_handle_t
typedef for flexio_mculcd_handle_t in advance.

typedef void (*flexio_mculcd_transfer_callback_t)(FLEXIO_MCULCD_Type *base,
flexio_mculcd_handle_t *handle, *status_t* status, void *userData)
FlexIO MCULCD callback for finished transfer.
When transfer finished, the callback function is called and returns the `status` as `kStatus_FLEXIO_MCULCD_Idle`.

FLEXIO_MCULCD_WAIT_COMPLETE_TIME
The delay time to wait for FLEXIO transmit complete.
Currently there is no method to detect whether the data has been sent out from the shifter, so the driver use a software delay for this. When the data is written to shifter buffer, the driver call the delay function to wait for the data shift out. If this value is too small, then the last few bytes might be lost when writing data using interrupt method or DMA method.

FLEXIO_MCULCD_DATA_BUS_WIDTH
The data bus width, must be 8 or 16.

FLEXIO_MCULCD_LEGACY_GPIO_FUNC
Whether to use legacy GPIO functions to control the CS/RS/RDWR pin signal.
If using the legacy pin functions, there is no user defined argument passed to the function.

struct *flexio_mculcd_type*
#include <fsl_flexio_mculcd.h> Define FlexIO MCULCD access structure typedef.

Public Members

`FLEXIO_Type *flexioBase`

FlexIO base pointer.

`flexio_mculcd_bus_t` busType

The bus type, 8080 or 6800.

`uint8_t dataPinStartIndex`

Start index of the data pin, the FlexIO pin `dataPinStartIndex` to (`dataPinStartIndex + FLEXIO_MCULCD_DATA_BUS_WIDTH - 1`) will be used for data transfer. Only support data bus width 8 and 16.

`uint8_t ENWRPIndex`

Pin select for WR(8080 mode), EN(6800 mode).

`uint8_t RDPinIndex`

Pin select for RD(8080 mode), not used in 6800 mode.

`uint8_t txShifterStartIndex`

Start index of shifters used for data write, it must be 0 or 4.

`uint8_t txShifterEndIndex`

End index of shifters used for data write.

`uint8_t rxShifterStartIndex`

Start index of shifters used for data read.

`uint8_t rxShifterEndIndex`

End index of shifters used for data read, it must be 3 or 7.

`uint8_t timerIndex`

Timer index used in FlexIO MCULCD.

`flexio_mculcd_pin_func_t setCSPin`

Function to set or clear the CS pin.

`flexio_mculcd_pin_func_t setRSPin`

Function to set or clear the RS pin.

`flexio_mculcd_pin_func_t setRDWRPIn`

Function to set or clear the RD/WR pin, only used in 6800 mode.

`struct _flexio_mculcd_config`

`#include <fsl_flexio_mculcd.h>` Define FlexIO MCULCD configuration structure.

Public Members

`bool enable`

Enable/disable FlexIO MCULCD after configuration.

`bool enableInDoze`

Enable/disable FlexIO operation in doze mode.

`bool enableInDebug`

Enable/disable FlexIO operation in debug mode.

`bool enableFastAccess`

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in bit-per-second for all data lines combined.

struct _flexio_mculcd_transfer

#include <fsl_flexio_mculcd.h> Define FlexIO MCULCD transfer structure.

Public Members

uint32_t command

Command to send.

uint32_t dataAddrOrSameValue

When sending the same value for many times, this is the value to send. When writing or reading array, this is the address of the data array.

size_t dataSize

How many bytes to transfer.

flexio_mculcd_transfer_mode_t mode

Transfer mode.

bool dataOnly

Send data only when tx without the command.

struct _flexio_mculcd_handle

#include <fsl_flexio_mculcd.h> Define FlexIO MCULCD handle structure.

Public Members

uint32_t dataAddrOrSameValue

When sending the same value for many times, this is the value to send. When writing or reading array, this is the address of the data array.

size_t dataCount

Total count to be transferred.

volatile size_t remainingCount

Remaining count to transfer.

volatile uint32_t state

FlexIO MCULCD internal state.

flexio_mculcd_transfer_callback_t completionCallback

FlexIO MCULCD transfer completed callback.

void *userData

Callback parameter.

2.25 FlexIO SPI Driver

void FLEXIO_SPI_MasterInit(*FLEXIO_SPI_Type* *base, *flexio_spi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_MasterGetDefaultConfig().

Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
    .enableMaster = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 500000,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $2*2=4$. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by $(1.5+2.5)*2=8$.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration.

The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_SlaveGetDefaultConfig().

Note: 1. Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3. For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $3 \times 2 = 6$. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by $(1.5 + 2.5) \times 2 = 8$.

Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0}
};
flexio_spi_slave_config_t config = {
    .enableSlave = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – status flag The parameter can be any combination of the following values:
 - kFLEXIO_SPI_TxEmptyFlag
 - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_DisableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source The parameter can be any combination of the following values:
 - kFLEXIO_SPI_RxFullInterruptEnable
 - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_EnableDMA(*FLEXIO_SPI_Type* *base, uint32_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(*FLEXIO_SPI_Type* *base,
flexio_spi_shift_direction_t
direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

```
static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(FLEXIO_SPI_Type *base,
                                                         flexio_spi_shift_direction_t
                                                         direction)
```

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

```
static inline void FLEXIO_SPI_Enable(FLEXIO_SPI_Type *base, bool enable)
```

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.
- enable – True to enable, false does not have any effect.

```
void FLEXIO_SPI_MasterSetBaudRate(FLEXIO_SPI_Type *base, uint32_t baudRate_Bps,
                                  uint32_t srcClockHz)
```

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

```
static inline void FLEXIO_SPI_WriteData(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t
                                         direction, uint32_t data)
```

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

```
static inline uint32_t FLEXIO_SPI_ReadData(FLEXIO_SPI_Type *base,  
                                           flexio_spi_shift_direction_t direction)
```

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

```
status_t FLEXIO_SPI_WriteBlocking(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                  direction, const uint8_t *buffer, size_t size)
```

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

```
status_t FLEXIO_SPI_ReadBlocking(FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t  
                                  direction, uint8_t *buffer, size_t size)
```

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the *FLEXIO_SPI_Type* structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base,
flexio_spi_master_handle_t *handle,
flexio_spi_master_transfer_callback_t
callback, void *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base,
flexio_spi_master_handle_t *handle,
flexio_spi_transfer_t *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

- kStatus_Success – Successfully start a transfer.

- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – SPI is not idle, is running another transfer.

`void FLEXIO_SPI_MasterTransferAbort(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle)`

Aborts the master data transfer, which used IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.

`status_t FLEXIO_SPI_MasterTransferGetCount(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, size_t *count)`

Gets the data transfer status which used IRQ.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

`void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)`

FlexIO SPI master IRQ handler function.

Parameters

- `spiType` – Pointer to the `FLEXIO_SPI_Type` structure.
- `spiHandle` – Pointer to the `flexio_spi_master_handle_t` structure to store the transfer state.

`status_t FLEXIO_SPI_SlaveTransferCreateHandle(FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, flexio_spi_slave_transfer_callback_t callback, void *userData)`

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- `base` – Pointer to the `FLEXIO_SPI_Type` structure.
- `handle` – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.


```
status_t FLEXIO_SPI_SlaveTransferNonBlocking(FLEXIO_SPI_Type *base,
                                             flexio_spi_slave_handle_t *handle,
                                             flexio_spi_transfer_t *xfer)
```

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- base – Pointer to the `FLEXIO_SPI_Type` structure.
- xfer – FlexIO SPI transfer structure. See `flexio_spi_transfer_t`.

Return values

- `kStatus_Success` – Successfully start a transfer.
- `kStatus_InvalidArgument` – Input argument is invalid.
- `kStatus_FLEXIO_SPI_Busy` – SPI is not idle; it is running another transfer.

```
static inline void FLEXIO_SPI_SlaveTransferAbort(FLEXIO_SPI_Type *base,
                                                flexio_spi_slave_handle_t *handle)
```

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the `FLEXIO_SPI_Type` structure.
- handle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
static inline status_t FLEXIO_SPI_SlaveTransferGetCount(FLEXIO_SPI_Type *base,
                                                       flexio_spi_slave_handle_t *handle,
                                                       size_t *count)
```

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the `FLEXIO_SPI_Type` structure.
- handle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

```
void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)
```

FlexIO SPI slave IRQ handler function.

Parameters

- spiType – Pointer to the `FLEXIO_SPI_Type` structure.
- spiHandle – Pointer to the `flexio_spi_slave_handle_t` structure to store the transfer state.

```
FSL_FLEXIO_SPI_DRIVER_VERSION
```

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator kStatus_FLEXIO_SPI_Busy
FlexIO SPI is busy.

enumerator kStatus_FLEXIO_SPI_Idle
SPI is idle

enumerator kStatus_FLEXIO_SPI_Error
FlexIO SPI error.

enumerator kStatus_FLEXIO_SPI_Timeout
FlexIO SPI timeout polling status flags.

enum _flexio_spi_clock_phase
FlexIO SPI clock phase configuration.

Values:

enumerator kFLEXIO_SPI_ClockPhaseFirstEdge
First edge on SPCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge
First edge on SPCK occurs at the start of the first cycle of a data transfer.

enum _flexio_spi_shift_direction
FlexIO SPI data shifter direction options.

Values:

enumerator kFLEXIO_SPI_MsbFirst
Data transfers start with most significant bit.

enumerator kFLEXIO_SPI_LsbFirst
Data transfers start with least significant bit.

enum _flexio_spi_data_bitcount_mode
FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode
8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags

Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag

Receive buffer full flag.

enum _flexio_spi_dma_enable

Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable

Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable

Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable

All DMA request source

enum _flexio_spi_transfer_flags

Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb

FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb

FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb

FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb

FlexIO SPI 16-bit LSB first

enumerator kFLEXIO_SPI_32bitMsb

FlexIO SPI 32-bit MSB first

enumerator kFLEXIO_SPI_32bitLsb

FlexIO SPI 32-bit LSB first

enumerator kFLEXIO_SPI_csContinuous

Enable the CS signal continuous mode

typedef enum _flexio_spi_clock_phase flexio_spi_clock_phase_t

FlexIO SPI clock phase configuration.

typedef enum _flexio_spi_shift_direction flexio_spi_shift_direction_t

FlexIO SPI data shifter direction options.

typedef enum _flexio_spi_data_bitcount_mode flexio_spi_data_bitcount_mode_t

FlexIO SPI data length mode options.

```
typedef struct _flexio_spi_type FLEXIO_SPI_Type
    Define FlexIO SPI access structure typedef.

typedef struct _flexio_spi_master_config flexio_spi_master_config_t
    Define FlexIO SPI master configuration structure.

typedef struct _flexio_spi_slave_config flexio_spi_slave_config_t
    Define FlexIO SPI slave configuration structure.

typedef struct _flexio_spi_transfer flexio_spi_transfer_t
    Define FlexIO SPI transfer structure.

typedef struct _flexio_spi_master_handle flexio_spi_master_handle_t
    typedef for flexio_spi_master_handle_t in advance.

typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t
    Slave handle is the same with master handle.

typedef void (*flexio_spi_master_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_master_handle_t *handle, status_t status, void *userData)
    FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_transfer_callback_t)(FLEXIO_SPI_Type *base,
flexio_spi_slave_handle_t *handle, status_t status, void *userData)
    FlexIO SPI slave callback for finished transmit.

FLEXIO_SPI_DUMMYDATA
    FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

SPI_RETRY_TIMES
    Retry times for waiting flag.

FLEXIO_SPI_XFER_DATA_FORMAT(flag)
    Get the transfer data format of width and bit order.

struct _flexio_spi_type
    #include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.
```

Public Members

```
FLEXIO_Type *flexioBase
    FlexIO base pointer.

uint8_t SDOPinIndex
    Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as
    GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex
    Pin select for data input.

uint8_t SCKPinIndex
    Pin select for clock.

uint8_t CSnPinIndex
    Pin select for enable.

uint8_t shifterIndex[2]
    Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]
    Timer index used in FlexIO SPI.
```

```
struct _flexio_spi_master_config
```

```
#include <fsl_flexio_spi.h> Define FlexIO SPI master configuration structure.
```

Public Members

```
bool enableMaster
```

Enable/disable FlexIO SPI master after configuration.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode.

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode.

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
uint32_t baudRate_Bps
```

Baud rate in Bps.

```
flexio_spi_clock_phase_t phase
```

Clock phase.

```
flexio_spi_data_bitcount_mode_t dataMode
```

8bit or 16bit mode.

```
struct _flexio_spi_slave_config
```

```
#include <fsl_flexio_spi.h> Define FlexIO SPI slave configuration structure.
```

Public Members

```
bool enableSlave
```

Enable/disable FlexIO SPI slave after configuration.

```
bool enableInDoze
```

Enable/disable FlexIO operation in doze mode.

```
bool enableInDebug
```

Enable/disable FlexIO operation in debug mode.

```
bool enableFastAccess
```

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

```
flexio_spi_clock_phase_t phase
```

Clock phase.

```
flexio_spi_data_bitcount_mode_t dataMode
```

8bit or 16bit mode.

```
struct _flexio_spi_transfer
```

```
#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.
```

Public Members

```
const uint8_t *txData
```

Send buffer.

```
uint8_t *rxData
    Receive buffer.

size_t dataSize
    Transfer bytes.

uint8_t flags
    FlexIO SPI control flag, MSB first or LSB first.

struct _flexio_spi_master_handle
    #include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.
```

Public Members

```
const uint8_t *txData
    Transfer buffer.

uint8_t *rxData
    Receive buffer.

size_t transferSize
    Total bytes to be transferred.

volatile size_t txRemainingBytes
    Send data remaining in bytes.

volatile size_t rxRemainingBytes
    Receive data remaining in bytes.

volatile uint32_t state
    FlexIO SPI internal state.

uint8_t bytePerFrame
    SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction
    Shift direction.

flexio_spi_master_transfer_callback_t callback
    FlexIO SPI callback.

void *userData
    Callback parameter.

bool isCsContinuous
    Is current transfer using CS continuous mode.

uint32_t timer1Cfg
    TIMER1 TIMCFG regiser value backup.
```

2.26 FlexIO UART Driver

```
status_t FLEXIO_UART_Init(FLEXIO_UART_Type *base, const flexio_uart_config_t *userConfig,
    uint32_t srcClock_Hz)
```

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_GetDefaultConfig().

Example

```

FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
    .bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);

```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXIO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```

flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);

```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

```
void FLEXIO_UART_ClearStatusFlags(FLEXIO_UART_Type *base, uint32_t mask)
```

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- mask – Status flag. The parameter can be any combination of the following values:
 - kFLEXIO_UART_TxDataRegEmptyFlag
 - kFLEXIO_UART_RxEmptyFlag
 - kFLEXIO_UART_RxOverRunFlag

```
void FLEXIO_UART_EnableInterrupts(FLEXIO_UART_Type *base, uint32_t mask)
```

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- mask – Interrupt source.

```
void FLEXIO_UART_DisableInterrupts(FLEXIO_UART_Type *base, uint32_t mask)
```

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- mask – Interrupt source.

```
static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(FLEXIO_UART_Type *base)
```

Gets the FlexIO UART transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.

Returns

FlexIO UART transmit data register address.

```
static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(FLEXIO_UART_Type *base)
```

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.

Returns

FlexIO UART receive data register address.

```
static inline void FLEXIO_UART_EnableTxDMA(FLEXIO_UART_Type *base, bool enable)
```

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_EnableRxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_Enable(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

static inline void FLEXIO_UART_WriteByte(*FLEXIO_UART_Type* *base, const uint8_t *buffer)

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

static inline void FLEXIO_UART_ReadByte(*FLEXIO_UART_Type* *base, uint8_t *buffer)

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.

- `kStatus_Success` – Successfully wrote all data.

`status_t FLEXIO_UART_ReadBlocking(FLEXIO_UART_Type *base, uint8_t *rxData, size_t rxSize)`

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- `base` – Pointer to the `FLEXIO_UART_Type` structure.
- `rxData` – The buffer to store the received bytes.
- `rxSize` – The number of data bytes to be received.

Return values

- `kStatus_FLEXIO_UART_Timeout` – Transmission timed out and was aborted.
- `kStatus_Success` – Successfully received all data.

`status_t FLEXIO_UART_TransferCreateHandle(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_callback_t callback, void *userData)`

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the `FLEXIO_UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing `NULL` as `ringBuffer`.

Parameters

- `base` – to `FLEXIO_UART_Type` structure.
- `handle` – Pointer to the `flexio_uart_handle_t` structure to store the transfer state.
- `callback` – The callback function.
- `userData` – The parameter of the callback function.

Return values

- `kStatus_Success` – Successfully create the handle.
- `kStatus_OutOfRange` – The FlexIO type/handle/ISR table out of range.

`void FLEXIO_UART_TransferStartRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)`

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn’t call the `UART_ReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

```
void FLEXIO_UART_TransferStopRingBuffer(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                         *handle)
```

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

```
status_t FLEXIO_UART_TransferSendNonBlocking(FLEXIO_UART_Type *base,
                                              flexio_uart_handle_t *handle,
                                              flexio_uart_transfer_t *xfer)
```

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See flexio_uart_transfer_t.

Return values

- kStatus_Success – Successfully starts the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished, data not written to the TX register.

```
void FLEXIO_UART_TransferAbortSend(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                   *handle)
```

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetSendCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *size_t* *count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferReceiveNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer, *size_t* *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter *kStatus_UART_RxIdle*. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to *xfer->data*. This function returns with the parameter *receivedBytes* set to 5. For the last 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to *xfer->data*. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See *flexio_uart_transfer_t*.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

```
void FLEXIO_UART_TransferAbortReceive(FLEXIO_UART_Type *base, flexio_uart_handle_t
                                     *handle)
```

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- handle – Pointer to the *flexio_uart_handle_t* structure to store the transfer state.

```
status_t FLEXIO_UART_TransferGetReceiveCount(FLEXIO_UART_Type *base,
                                              flexio_uart_handle_t *handle, size_t *count)
```

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.
- handle – Pointer to the *flexio_uart_handle_t* structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- *kStatus_NoTransferInProgress* – transfer has finished or no transfer in progress.
- *kStatus_Success* – Successfully return the count.

```
void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)
```

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the *FLEXIO_UART_Type* structure.
- uartHandle – Pointer to the *flexio_uart_handle_t* structure to store the transfer state.

```
void FLEXIO_UART_FlushShifters(FLEXIO_UART_Type *base)
```

Flush tx/rx shifters.

Parameters

- base – Pointer to the *FLEXIO_UART_Type* structure.

```
FSL_FLEXIO_UART_DRIVER_VERSION
```

FlexIO UART driver version.

Error codes for the UART driver.

Values:

```
enumerator kStatus_FLEXIO_UART_TxBusy
```

Transmitter is busy.

```
enumerator kStatus_FLEXIO_UART_RxBusy
```

Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle

UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR

ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverflow

UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverflow

UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout

UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport

Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char

FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar

7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar

8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar

9-bit data characters

enum _flexio_uart_interrupt_enable

Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable

Receive buffer full interrupt enable.

enum _flexio_uart_status_flags

Define FlexIO UART status mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag

Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag

Receive buffer over run flag.

typedef enum _flexio_uart_bit_count_per_char flexio_uart_bit_count_per_char_t

FlexIO UART bit count per char.

```
typedef struct _flexio_uart_type FLEXIO_UART_Type
    Define FlexIO UART access structure typedef.
typedef struct _flexio_uart_config flexio_uart_config_t
    Define FlexIO UART user configuration structure.
typedef struct _flexio_uart_transfer flexio_uart_transfer_t
    Define FlexIO UART transfer structure.
typedef struct _flexio_uart_handle flexio_uart_handle_t
typedef void (*flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t
*handle, status_t status, void *userData)
    FlexIO UART transfer callback function.
UART_RETRY_TIMES
    Retry times for waiting flag.
struct _flexio_uart_type
    #include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.
```

Public Members

```
FLEXIO_Type *flexioBase
    FlexIO base pointer.
uint8_t TxPinIndex
    Pin select for UART_Tx.
uint8_t RxPinIndex
    Pin select for UART_Rx.
uint8_t shifterIndex[2]
    Shifter index used in FlexIO UART.
uint8_t timerIndex[2]
    Timer index used in FlexIO UART.
struct _flexio_uart_config
    #include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.
```

Public Members

```
bool enableUart
    Enable/disable FlexIO UART TX & RX.
bool enableInDoze
    Enable/disable FlexIO operation in doze mode
bool enableInDebug
    Enable/disable FlexIO operation in debug mode
bool enableFastAccess
    Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to
    be at least twice the frequency of the bus clock.
uint32_t baudRate_Bps
    Baud rate in Bps.
```

flexio_uart_bit_count_per_char_t bitCountPerChar
number of bits, 7/8/9 -bit

struct *_flexio_uart_transfer*

#include <fsl_flexio_uart.h> Define FlexIO UART transfer structure.

Public Members

size_t dataSize

Transfer size

struct *_flexio_uart_handle*

#include <fsl_flexio_uart.h> Define FLEXIO UART handle structure.

Public Members

const uint8_t *volatile txData

Address of remaining data to send.

volatile size_t txDataSize

Size of the remaining data to send.

uint8_t *volatile rxData

Address of remaining data to receive.

volatile size_t rxDataSize

Size of the remaining data to receive.

size_t txDataSizeAll

Total bytes to be sent.

size_t rxDataSizeAll

Total bytes to be received.

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

size_t rxRingBufferSize

Size of the ring buffer.

volatile uint16_t rxRingBufferHead

Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail

Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

union *__unnamed84__*

Public Members

uint8_t *data

The buffer of data to be transfer.

uint8_t *rxData

The buffer to receive data.

const uint8_t *txData

The buffer of data to be sent.

2.27 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32_t FLEXSPI_GetInstance(FLEXSPI_Type *base)

Get the instance number for FLEXSPI.

Parameters

- base – FLEXSPI base pointer.

status_t FLEXSPI_CheckAndClearError(FLEXSPI_Type *base, uint32_t status)

Check and clear IP command execution errors.

Parameters

- base – FLEXSPI base pointer.
- status – interrupt status.

void FLEXSPI_Init(FLEXSPI_Type *base, const flexspi_config_t *config)

Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI_GetDefaultConfig(flexspi_config_t *config)

Gets default settings for FLEXSPI.

Parameters

- config – FLEXSPI configuration structure.

void FLEXSPI_Deinit(FLEXSPI_Type *base)

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

- base – FLEXSPI peripheral base address.

void FLEXSPI_UpdateDllValue(FLEXSPI_Type *base, flexspi_device_config_t *config, flexspi_port_t port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.

- port – FLEXSPI Operation port.

```
void FLEXSPI_SetFlashConfig(FLEXSPI_Type *base, flexspi_device_config_t *config,  
                           flexspi_port_t port)
```

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

```
void FLEXSPI_SoftwareReset(FLEXSPI_Type *base)
```

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – FLEXSPI peripheral base address.

```
static inline void FLEXSPI_Enable(FLEXSPI_Type *base, bool enable)
```

Enables or disables the FLEXSPI module.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

```
void FLEXSPI_UpdateAhbBuffersSettings(FLEXSPI_Type *base, flexspi_ahbBuffers_ctrl_t  
                                     *ptrAhbBufferCtrl)
```

Update all AHB buffers' settings, including buffer size, master ID.

Parameters

- base – FLEXSPI peripheral base address.
- ptrAhbBufferCtrl – Pointer to structure flexspi_ahbBuffers_ctrl_t which store all AHB buffers' settings.

```
static inline void FLEXSPI_EnableInterrupts(FLEXSPI_Type *base, uint32_t mask)
```

Enables the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)
```

Disable the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)
```

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP tx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – tx fifo address.

static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP rx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – rx fifo address.

static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)

Clears the FLEXSPI IP FIFO logic.

Parameters

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)

Gets the valid data entries in the FLEXSPI FIFOs.

Parameters

- base – FLEXSPI peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.

Return values

interrupt – status flag, use status flag to AND flexspi_flags_t could get the related status.

```
static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)
```

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

```
static inline flexspi_arb_command_source_t FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type *base)
```

Gets the trigger source of current command sequence granted by arbitrator.

Parameters

- base – FLEXSPI peripheral base address.

Return values

trigger – source of current command sequence.

```
static inline flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)
```

Gets the error code when IP command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when IP command error detected.

```
static inline flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)
```

Gets the error code when AHB command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when AHB command error detected.

```
static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)
```

Returns whether the bus is idle.

Parameters

- base – FLEXSPI peripheral base address.

Return values

- true – Bus is idle.
- false – Bus is busy.

```
void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, flexspi_read_sample_clock_t clockSource)
```

Update read sample clock source.

Parameters

- base – FLEXSPI peripheral base address.

- clockSource – clockSource of type flexspi_read_sample_clock_t

void FLEXSPI_UpdateLUT(FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t count)

Updates the LUT table.

Parameters

- base – FLEXSPI peripheral base address.
- index – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory.
- cmd – Command sequence array.
- count – Number of sequences.

static inline void FLEXSPI_WriteData(FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)

Writes data into FIFO.

Parameters

- base – FLEXSPI peripheral base address
- data – The data bytes to send
- fifoIndex – Destination fifo index.

static inline uint32_t FLEXSPI_ReadData(FLEXSPI_Type *base, uint8_t fifoIndex)

Receives data from data FIFO.

Parameters

- base – FLEXSPI peripheral base address
- fifoIndex – Source fifo index.

Returns

The data in the FIFO.

status_t FLEXSPI_WriteBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to send

Return values

- kStatus_Success – write success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_ReadBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to receive

Return values

- kStatus_Success – read success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_TransferBlocking(FLEXSPI_Type *base, *flexspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Parameters

- base – FLEXSPI peripheral base address
- xfer – pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_callback_t* callback, void *userData)

Initializes the FLEXSPI handle which is used in transactional functions.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to *flexspi_handle_t* structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_t* *xfer)

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call `FLEXSPI_GetTransferCount` to poll the transfer status to check whether the transfer is finished. If the return status is not `kStatus_FLEXSPI_Busy`, the transfer is finished. For `FLEXSPI_Read`, the `dataSize` should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `xfer` – pointer to `flexspi_transfer_t` structure.

Return values

- `kStatus_Success` – Successfully start the data transmission.
- `kStatus_FLEXSPI_Busy` – Previous transmission still not finished.

`status_t` `FLEXSPI_TransferGetCount(FLEXSPI_Type *base, flexspi_handle_t *handle, size_t *count)`

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state.
- `count` – Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_InvalidArgument` – count is Invalid.
- `kStatus_Success` – Successfully return the count.

`void` `FLEXSPI_TransferAbort(FLEXSPI_Type *base, flexspi_handle_t *handle)`

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure which stores the transfer state

`void` `FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, flexspi_handle_t *handle)`

Master interrupt handler.

Parameters

- `base` – FLEXSPI peripheral base address.
- `handle` – pointer to `flexspi_handle_t` structure.

`FSL_FLEXSPI_DRIVER_VERSION`

FLEXSPI driver version.

Status structure of FLEXSPI.

Values:

enumerator kStatus_FLEXSPI_Busy

FLEXSPI is busy

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandSequenceError

IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandGrantTimeout

IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, `_flexspi_command`.

Values:

enumerator kFLEXSPI_Command_STOP

Stop execution, deassert CS.

enumerator kFLEXSPI_Command_SDR

Transmit Command code to Flash, using SDR mode.

enumerator kFLEXSPI_Command_RADDR_SDR

Transmit Row Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_CADDR_SDR

Transmit Column Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE1_SDR

Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE2_SDR

Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE4_SDR

Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE8_SDR

Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_WRITE_SDR

Transmit Programming Data to Flash, using SDR mode.

enumerator kFLEXSPI_Command_READ_SDR

Receive Read Data from Flash, using SDR mode.

enumerator kFLEXSPI_Command_LEARN_SDR

Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator kFLEXSPI_Command_DATSZ_SDR

Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator kFLEXSPI_Command_DUMMY_SDR

Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_SDR

Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_DDR

Transmit Command code to Flash, using DDR mode.

enumerator kFLEXSPI_Command_RADDR_DDR

Transmit Row Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_CADDR_DDR

Transmit Column Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE1_DDR

Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE2_DDR

Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE4_DDR

Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE8_DDR

Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_WRITE_DDR

Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI_Command_READ_DDR

Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI_Command_LEARN_DDR

Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI_Command_DATSZ_DDR

Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI_Command_DUMMY_DDR

Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_DDR

Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_JUMP_ON_CS

Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum _flexspi_pad

pad definition of FLEXSPI, use to form LUT instruction.

Values:

enumerator kFLEXSPI_1PAD

Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI_2PAD

Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI_4PAD

Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI_8PAD

Transmit command/address and transmit/receive data only through DATA[7:0].

enum _flexspi_flags

FLEXSPI interrupt status flags.

Values:

enumerator kFLEXSPI_SequenceExecutionTimeoutFlag

Sequence execution timeout.

enumerator kFLEXSPI_AhbBusErrorFlag

AHB Bus error flag.

enumerator kFLEXSPI_SckStoppedBecauseTxEmptyFlag

SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI_SckStoppedBecauseRxFullFlag

SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI_IpTxFifoWatermarkEmptyFlag

IP TX FIFO WaterMark empty.

enumerator kFLEXSPI_IpRxFifoWatermarkAvailableFlag

IP RX FIFO WaterMark available.

enumerator kFLEXSPI_AhbCommandSequenceErrorFlag

AHB triggered Command Sequences Error.

enumerator kFLEXSPI_IpCommandSequenceErrorFlag

IP triggered Command Sequences Error.

enumerator kFLEXSPI_AhbCommandGrantTimeoutFlag

AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandGrantTimeoutFlag

IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandExecutionDoneFlag

IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI_AllInterruptFlags

All flags.

enum _flexspi_read_sample_clock

FLEXSPI sample clock source selection for Flash Reading.

Values:

enumerator kFLEXSPI_ReadSampleClkLoopbackInternally

Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromDqsPad

Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromSckPad

SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI_ReadSampleClkExternalInputFromDqsPad

Flash provided Read strobe and input from DQS pad.

enum _flexspi_cs_interval_cycle_unit

FLEXSPI interval unit for flash device select.

Values:

enumerator kFLEXSPI_CsIntervalUnit1SckCycle

Chip selection interval: CSINTERVAL * 1 serial clock cycle.

enumerator kFLEXSPI_CsIntervalUnit256SckCycle

Chip selection interval: CSINTERVAL * 256 serial clock cycle.

enum _flexspi_ahb_write_wait_unit

FLEXSPI AHB wait interval unit for writing.

Values:

enumerator kFLEXSPI_AhbWriteWaitUnit2AhbCycle

AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8AhbCycle

AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32AhbCycle

AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit128AhbCycle

AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit512AhbCycle

AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit2048AhbCycle

AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8192AhbCycle

AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32768AhbCycle

AWRWAIT unit is 32768 ahb clock cycle.

enum _flexspi_ip_error_code

Error Code when IP command Error detected.

Values:

enumerator kFLEXSPI_IpCmdErrorNoError

No error.

enumerator kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd

IP command with JMP_ON_CS instruction used.

enumerator kFLEXSPI_IpCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_IpCmdErrorInvalidAddress

Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI_IpCmdErrorSequenceExecutionTimeout

Sequence execution timeout.

enumerator kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss

Flash boundary crossed.

enum _flexspi_ahb_error_code

Error Code when AHB command Error detected.

Values:

enumerator kFLEXSPI_AhbCmdErrorNoError

No error.

enumerator kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd

AHB Write command with JMP_ON_CS instruction used in the sequence.

enumerator kFLEXSPI_AhbCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_AhbCmdSequenceExecutionTimeout

Sequence execution timeout.

enum _flexspi_port

FLEXSPI operation port select.

Values:

enumerator kFLEXSPI_PortA1

Access flash on A1 port.

enumerator kFLEXSPI_PortA2

Access flash on A2 port.

enumerator kFLEXSPI_PortCount

enum _flexspi_arb_command_source

Trigger source of current command sequence granted by arbitrator.

Values:

enumerator kFLEXSPI_AhbReadCommand

enumerator kFLEXSPI_AhbWriteCommand

enumerator kFLEXSPI_IpCommand

enumerator kFLEXSPI_SuspendedCommand

enum _flexspi_command_type

Command type.

Values:

enumerator kFLEXSPI_Command

FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI_Config

FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI_Read

enumerator kFLEXSPI_Write

typedef enum _flexspi_pad flexspi_pad_t

pad definition of FLEXSPI, use to form LUT instruction.

typedef enum _flexspi_flags flexspi_flags_t

FLEXSPI interrupt status flags.

```

typedef enum _flexspi_read_sample_clock flexspi_read_sample_clock_t
    FLEXSPI sample clock source selection for Flash Reading.

typedef enum _flexspi_cs_interval_cycle_unit flexspi_cs_interval_cycle_unit_t
    FLEXSPI interval unit for flash device select.

typedef enum _flexspi_ahb_write_wait_unit flexspi_ahb_write_wait_unit_t
    FLEXSPI AHB wait interval unit for writing.

typedef enum _flexspi_ip_error_code flexspi_ip_error_code_t
    Error Code when IP command Error detected.

typedef enum _flexspi_ahb_error_code flexspi_ahb_error_code_t
    Error Code when AHB command Error detected.

typedef enum _flexspi_port flexspi_port_t
    FLEXSPI operation port select.

typedef enum _flexspi_arb_command_source flexspi_arb_command_source_t
    Trigger source of current command sequence granted by arbitrator.

typedef enum _flexspi_command_type flexspi_command_type_t
    Command type.

typedef struct _flexspi_ahbBuffer_config flexspi_ahbBuffer_config_t

typedef struct _flexspi_ahbBuffers_ctrl flexspi_ahbBuffers_ctrl_t
    Structure to control all AHB buffers.

typedef struct _flexspi_config flexspi_config_t
    FLEXSPI configuration structure.

typedef struct _flexspi_device_config flexspi_device_config_t
    External device configuration items.

typedef struct _flexspi_transfer flexspi_transfer_t
    Transfer structure for FLEXSPI.

typedef struct _flexspi_handle flexspi_handle_t

typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, flexspi_handle_t *handle,
status_t status, void *userData)
    FLEXSPI transfer callback function.

typedef struct _flexspi_addr_map_config flexspi_addr_map_config_t
    Address mapping configuration structure.

FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT

FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
    Formula to form FLEXSPI instructions in LUT table.

struct _flexspi_ahbBuffer_config
    #include <fsl_flexspi.h>

```

Public Members

uint8_t priority

This priority for AHB Master Read which this AHB RX Buffer is assigned.

uint8_t masterIndex

AHB Master ID the AHB RX Buffer is assigned.

uint16_t bufferSize
AHB buffer size in byte.

bool enablePrefetch
AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows prefetch disable/enable separately for each master.

struct __flexspi_ahbBuffers_ctrl
#include <fsl_flexspi.h> Structure to control all AHB buffers.

Public Members

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]
Configurations of all AHB buffers.

struct __flexspi_config
#include <fsl_flexspi.h> FLEXSPI configuration structure.

Public Members

uint8_t clockDiv
FLEXSPI serial root clock divider.

flexspi_read_sample_clock_t rxSampleClock
Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning
Enable/disable SCK output free-running.

bool enableDoze
Enable/disable doze mode support.

bool enableHalfSpeedAccess
Enable/disable divide by 2 of the clock for half speed commands.

flexspi_read_sample_clock_t rxSampleClockPortB
Sample Clock source_b selection for Flash Reading.

bool rxSampleClockDiff
Sample Clock source or source_b selection for Flash Reading.

bool enableSameConfigForAll
Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16_t seqTimeoutCycle
Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCycle*1024 serial root clock cycles.

uint8_t ipGrantTimeoutCycle
Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle*1024 AHB clock cycles.

uint8_t txWatermark
FLEXSPI IP transmit watermark value.

uint8_t rxWatermark
FLEXSPI receive watermark value.

struct __flexspi_device_config
#include <fsl_flexspi.h> External device configuration items.

Public Members

`uint32_t flexspiRootClk`
FLEXSPI serial root clock.

`bool isSck2Enabled`
FLEXSPI use SCK2.

`uint32_t flashSize`
Flash size in KByte.

`bool addressShift`
Address shift.

flexspi_cs_interval_cycle_unit_t CSIntervalUnit
CS interval unit, 1 or 256 cycle.

`uint16_t CSInterval`
CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

`uint8_t CSHoldTime`
CS line hold time.

`uint8_t CSSetupTime`
CS line setup time.

`uint8_t dataValidTime`
Data valid time for external device.

`uint8_t columnSpace`
Column space size.

`bool enableWordAddress`
If enable word address.

`uint8_t AWRSeqIndex`
Sequence ID for AHB write command.

`uint8_t AWRSeqNumber`
Sequence number for AHB write command.

`uint8_t ARDSeqIndex`
Sequence ID for AHB read command.

`uint8_t ARDSeqNumber`
Sequence number for AHB read command.

flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit
AHB write wait unit.

`uint16_t AHBWriteWaitInterval`
AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

`bool enableWriteMask`
Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

`bool isFroClockSource`
Is FRO clock source or not.

`struct _flexspi_transfer`
#include <fsl_flexspi.h> Transfer structure for FLEXSPI.

Public Members

uint32_t deviceAddress

Operation device address.

flexspi_port_t port

Operation port.

flexspi_command_type_t cmdType

Execution command type.

uint8_t seqIndex

Sequence ID for command.

uint8_t SeqNumber

Sequence number for command.

uint32_t *data

Data buffer.

size_t dataSize

Data size in bytes.

struct *_flexspi_handle*

#include <fsl_flexspi.h> Transfer handle structure for FLEXSPI.

Public Members

uint32_t state

Internal state for FLEXSPI transfer

uint8_t *data

Data buffer.

size_t dataSize

Remaining Data size in bytes.

size_t transferTotalSize

Total Data size in bytes.

flexspi_transfer_callback_t completionCallback

Callback for users while transfer finish or error occurred

void *userData

FLEXSPI callback function parameter.

struct *_flexspi_addr_map_config*

#include <fsl_flexspi.h> Address mapping configuration structure.

Public Members

uint32_t addrStart

Remapping start address.

uint32_t addrEnd

Remapping end address.

uint32_t addrOffset

Address offset.

bool remapEnable
Enable address remapping.

struct ahbConfig

Public Members

uint8_t ahbGrantTimeoutCycle
Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCyle*1024 AHB clock cycles.

uint16_t ahbBusTimeoutCycle
Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle*1024 AHB clock cycles.

uint8_t resumeWaitCycle
Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

bool disableAhbReadResume
True: Suspended AHB read prefetch does not resume once aborted; **False:** Suspended AHB read prefetch resumes when AHB is IDLE.

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]
AHB buffer size.

bool enableClearAHBBufferOpt
Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt
Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch
Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable
Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable
Enable AHB bus cachable read access support.

2.28 FLEXSPI eDMA Driver

void FLEXSPI_TransferCreateHandleEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle, flexspi_edma_callback_t callback, void *userData, edma_handle_t *txDmaHandle, edma_handle_t *rxDmaHandle)

Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_edma_handle_t structure

- callback – FLEXSPI callback, NULL means no callback.
- userData – User callback function data.
- txDmaHandle – User requested DMA handle for TX DMA transfer.
- rxDmaHandle – User requested DMA handle for RX DMA transfer.

void FLEXSPI_TransferUpdateSizeEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle, flexspi_edma_transfer_nsize_t nsize)

Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

See also:

flexspi_edma_transfer_nsize_t .

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_edma_handle_t structure
- nsize – FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXSPI_EDMASize1Bytes(one byte).

status_t FLEXSPI_TransferEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle, flexspi_transfer_t *xfer)

Transfers FLEXSPI data using an eDMA non-blocking method.

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure
- xfer – FLEXSPI transfer structure.

Return values

- kStatus_FLEXSPI_Busy – FLEXSPI is busy transfer.
- kStatus_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer.
- kStatus_Success – FLEXSPI successfully start edma transfer.

void FLEXSPI_TransferAbortEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle)

Aborts the transfer data using eDMA.

This function aborts the transfer data using eDMA.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure

status_t FLEXSPI_TransferGetTransferCountEDMA(FLEXSPI_Type *base, flexspi_edma_handle_t *handle, size_t *count)

Gets the transferred counts of transfer.

Parameters

- base – FLEXSPI peripheral base address.

- `handle` – Pointer to `flexspi_edma_handle_t` structure.
- `count` – Bytes transfer.

Return values

- `kStatus_Success` – Succeed get the transfer count.
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`FSL_FLEXSPI_EDMA_DRIVER_VERSION`

FLEXSPI EDMA driver version.

`enum _flexspi_edma_ntransfer_size`

eDMA transfer configuration

Values:

enumerator `kFLEXPSI_EDMAAnSize1Bytes`

Source/Destination data transfer size is 1 byte every time

enumerator `kFLEXPSI_EDMAAnSize2Bytes`

Source/Destination data transfer size is 2 bytes every time

enumerator `kFLEXPSI_EDMAAnSize4Bytes`

Source/Destination data transfer size is 4 bytes every time

enumerator `kFLEXPSI_EDMAAnSize8Bytes`

Source/Destination data transfer size is 8 bytes every time

enumerator `kFLEXPSI_EDMAAnSize32Bytes`

Source/Destination data transfer size is 32 bytes every time

`typedef struct _flexspi_edma_handle flexspi_edma_handle_t`

`typedef void (*flexspi_edma_callback_t)(FLEXSPI_Type *base, flexspi_edma_handle_t *handle, status_t status, void *userData)`

FLEXSPI eDMA transfer callback function for finish and error.

`typedef enum _flexspi_edma_ntransfer_size flexspi_edma_transfer_nsize_t`

eDMA transfer configuration

`struct _flexspi_edma_handle`

#include <fsl_flexspi_edma.h> FLEXSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

`edma_handle_t *txDmaHandle`

eDMA handler for FLEXSPI Tx.

`edma_handle_t *rxDmaHandle`

eDMA handler for FLEXSPI Rx.

`size_t transferSize`

Bytes need to transfer.

`flexspi_edma_transfer_nsize_t nsize`

eDMA SSIZE/DSIZE in each transfer.

`uint8_t nbytes`

eDMA minor byte transfer count initially configured.

uint8_t count

The transfer data count in a DMA request.

uint32_t state

Internal state for FLEXSPI eDMA transfer.

flexspi_edma_callback_t completionCallback

A callback function called after the eDMA transfer is finished.

void *userData

User callback parameter

2.29 FREQME: Frequency Measurement

2.30 GLIKEY

2.31 GLIKEY

Values:

enumerator kStatus_GLIKEY_LockedError

GLIKEY status for locked SFR registers (unexpected) .

enumerator kStatus_GLIKEY_NotLocked

GLIKEY status for unlocked SFR registers.

enumerator kStatus_GLIKEY_Locked

GLIKEY status for locked SFR registers.

enumerator kStatus_GLIKEY_DisabledError

GLIKEY status for disabled error.

FSL_GLIKEY_DRIVER_VERSION

Defines GLIKEY driver version 2.0.1.

Change log:

- Version 2.0.1
 - Implement INIT state recovery from the LOCKED state after a reset when the previous index was locked.
- Version 2.0.0
 - Initial version

GLIKEY_CODEWORD_STEP1

GLIKEY_CODEWORD_STEP2

GLIKEY_CODEWORD_STEP3

GLIKEY_CODEWORD_STEP4

GLIKEY_CODEWORD_STEP5

GLIKEY_CODEWORD_STEP6

GLIKEY_CODEWORD_STEP7

GLIKEY_CODEWORD_STEP_EN

GLIKEY_FSM_WR_DIS

GLIKEY_FSM_INIT

GLIKEY_FSM_STEP1

GLIKEY_FSM_STEP2

GLIKEY_FSM_STEP3

GLIKEY_FSM_STEP4

GLIKEY_FSM_LOCKED

GLIKEY_FSM_WR_EN

GLIKEY_FSM_SSR_RESET

uint32_t GLIKEY_GetStatus(GLIKEY_Type *base)

Retreives the current status of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Glikey status information

status_t GLIKEY_IsLocked(GLIKEY_Type *base)

Get if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_Locked – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status

status_t GLIKEY_CheckLock(GLIKEY_Type *base)

Check if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_LockedError – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status kStatus_Success if success

status_t GLIKEY_SyncReset(GLIKEY_Type *base)

Perform a synchronous reset of Glikey.

This function performs a synchronounous reset of the Glikey. This results in:

- Glikey will return to the INIT state, unless it is in the LOCK state

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntEnable(GLIKEY_Type *base, uint32_t value)

Set interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[in]** Value to set the interrupt enable flag to, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_GetIntEnable(GLIKEY_Type *base, uint32_t *value)

Get interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[out]** Pointer which will be filled with the interrupt enable status, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success

status_t GLIKEY_ClearIntStatus(GLIKEY_Type *base)

Clear the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntStatus(GLIKEY_Type *base)

Set the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_Lock(GLIKEY_Type *base)

Lock Glikey SFR (Special Function Registers) interface.

This operation locks the Glikey SFR interface if it is not locked yet.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success

status_t GLIKEY_LockIndex(GLIKEY_Type *base)

Lock Glikey index.

This operation is used to lock a Glikey index. It can only be executed from the WR_EN state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kSta-
tus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_IsIndexLocked(GLIKEY_Type *base, uint32_t index)

Check if Glikey index is locked.

This operation returns the locking status of Glikey index.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns

kStatus_GLIKEY_Locked if locked, kStatus_GLIKEY_NotLocked if unlocked
Possible errors: kStatus_Fail

status_t GLIKEY_StartEnable(GLIKEY_Type *base, uint32_t index)

Start Glikey enable.

This operation is used to set a new index and start a the sequence to enable it. It needs to be started from the INIT state. If the new index is already locked Glikey will go to LOCKED state, otherwise it will go to STEP1 state. If this operation is used when Glikey is in any state other than INIT Glikey will go to WR_DIS state. It can only recover from this state through a reset (synchronous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kSta-
tus_GLIKEY_LockedError, kStatus_Fail

status_t GLIKEY_ContinueEnable(GLIKEY_Type *base, uint32_t codeword)

Continue Glikey enable.

This operation is used to progress through the different states of the state machine, starting from STEP1 until the state WR_EN is reached. Each next state of the state machine can only be reached by providing the right codeword to this function. If anything goes wrong the state machine will go to WR_DIS state and can only recover from it through a reset (synchronous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- codeword – **[in]** Encoded word for progressing to next FSM state (see GLIKEY_CODEWORD_STEPx/EN)

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_Fail, kStatus_GLIKEY_DisabledError

status_t GLIKEY_EndOperation(GLIKEY_Type *base)

End Glikey operation.

This operation is used to end a Glikey operation. It can only be executed from the WR_EN, LOCKED and RESET states. Executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. After this operation Glikey will go to INIT state or stay in LOCKED state when the index was locked. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

A code-flow protected error code (see nxpCsslFlowProtection)

Returns

Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_ResetIndex(GLIKEY_Type *base, uint32_t index)

Reset Glikey index.

This operation is used to reset a Glikey index. It can only be executed from the INIT state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchronous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active or the index is locked this operation will return an error.

Returns

A code-flow protected error code (see nxpCsslFlowProtection)

Returns

Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

2.32 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.

2.33 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,
```

(continues on next page)

(continued from previous page)

```
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)

2.34 I3C: I3C Driver

FSL_I3C_DRIVER_VERSION

I3C driver version.

I3C status return codes.

Values:

enumerator kStatus_I3C_Busy

The master is already performing a transfer.

enumerator kStatus_I3C_Idle

The slave driver is idle.

enumerator kStatus_I3C_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus_I3C_WriteAbort

The slave device sent a NAK in response to a write.

enumerator kStatus_I3C_Term

The master terminates slave read.

enumerator kStatus_I3C_HdrParityError

Parity error from DDR read.

enumerator kStatus_I3C_CrcError

CRC error from DDR read.

enumerator kStatus_I3C_ReadFifoError

Read from M/SRDATAB register when FIFO empty.

enumerator kStatus_I3C_WriteFifoError

Write to M/SWDATAB register when FIFO full.

enumerator kStatus_I3C_MsgError

Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus_I3C_InvalidReq

Invalid use of request.

enumerator kStatus_I3C_Timeout

The module has stalled too long in a frame.

enumerator kStatus_I3C_SlaveCountExceed

The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

enumerator kStatus_I3C_IBIWon

The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus_I3C_OverrunError

Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus_I3C_UnderrunError

Slave internal to-bus buffer/FIFO underrun

enumerator kStatus_I3C_UnderrunNak

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus_I3C_InvalidStart

Slave invalid start flag

enumerator kStatus_I3C_SdrParityError

SDR parity error

enumerator kStatus_I3C_S0S1Error

S0 or S1 error

enum _i3c_hdr_mode

I3C HDR modes.

Values:

enumerator kI3C_HDRModeNone

enumerator kI3C_HDRModeDDR

enumerator kI3C_HDRModeTSP

enumerator kI3C_HDRModeTSL

typedef enum _i3c_hdr_mode i3c_hdr_mode_t

I3C HDR modes.

typedef struct _i3c_device_info i3c_device_info_t

I3C device information.

I3C_RETRY_TIMES

Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C_MAX_DEVCNT

I3C_IBI_BUFF_SIZE

struct _i3c_device_info

#include <fsl_i3c.h> I3C device information.

Public Members

uint8_t dynamicAddr

Device dynamic address.

uint8_t staticAddr

Static address.

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint16_t maxReadLength

Maximum read length.

uint16_t maxWriteLength

Maximum write length.

uint8_t hdrMode

Support hdr mode, could be OR logic in i3c_hdr_mode.

2.35 I3C Common Driver

typedef struct *i3c_config* i3c_config_t

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

uint32_t I3C_GetInstance(I3C_Type *base)

Get which instance current I3C is used.

Parameters

- base – The I3C peripheral base address.

void I3C_GetDefaultConfig(*i3c_config_t* *config)

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```

config->enableMaster      = kI3C_MasterCapable;
config->disableTimeout    = false;
config->hKeep              = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop = true;
config->enableOpenDrainHigh = true;
config->baudRate_Hz.i2cBaud = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;

```

(continues on next page)

(continued from previous page)

```

config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress      = 0x0AU;
config->slowClock_Hz              = 1000000U;
config->enableSlave                = true;
config->vendorID                  = 0x11BU;
config->enableRandomPart          = false;
config->partNumber                = 0;
config->dcr                      = 0;
config->bcr = 0;
config->hdrMode                   = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest              = false;
config->ignoreS0S1Error          = false;
config->offline                   = false;
config->matchSlaveStartStop      = false;

```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with `I3C_Init()`.

Parameters

- `config` – **[out]** User provided configuration structure for default values. Refer to `i3c_config_t`.

`void I3C_Init(I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)`

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- `base` – The I3C peripheral base address.
- `config` – User provided peripheral configuration. Use `I3C_GetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

`struct _i3c_config`

`#include <fsl_i3c.h>` Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_GetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

`i3c_master_enable_t enableMaster`

Enable master mode.

`bool disableTimeout`

Whether to disable timeout to prevent the ERRWARN.

`i3c_master_hkeep_t hKeep`

High keeper mode setting.

`bool enableOpenDrainStop`

Whether to emit open-drain speed STOP.

`bool enableOpenDrainHigh`
Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

`i3c_baudrate_hz_t baudRate_Hz`
Desired baud rate settings.

`i3c_start_scl_delay_t startSclDelay`
I3C SCL delay after START.

`i3c_start_scl_delay_t restartSclDelay`
I3C SCL delay after Repeated START.

`uint8_t masterDynamicAddress`
Main master dynamic address configuration.

`uint32_t maxWriteLength`
Maximum write length.

`uint32_t maxReadLength`
Maximum read length.

`bool enableSlave`
Whether to enable slave.

`uint8_t staticAddr`
Static address.

`uint16_t vendorID`
Device vendor ID(manufacture ID).

`uint32_t partNumber`
Device part number info

`uint8_t dcr`
Device characteristics register information.

`uint8_t bcr`
Bus characteristics register information.

`uint8_t hdrMode`
Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.

`bool nakAllRequest`
Whether to reply NAK to all requests except broadcast CCC.

`bool ignoreS0S1Error`
Whether to ignore S0/S1 error in SDR mode.

`bool offline`
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

`bool matchSlaveStartStop`
Whether to assert start/stop status only the time slave is addressed.

2.36 I3C Master Driver

```
void I3C_MasterGetDefaultConfig(i3c_master_config_t *masterConfig)
```

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster      = kI3C_MasterOn;
masterConfig->disableTimeout    = false;
masterConfig->hKeep              = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop = true;
masterConfig->enableOpenDrainHigh = true;
masterConfig->baudRate_Hz        = 100000U;
masterConfig->busType            = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I3C_MasterInit()`.

Parameters

- `masterConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_master_config_t`.

```
void I3C_MasterInit(I3C_Type *base, const i3c_master_config_t *masterConfig, uint32_t
                    sourceClock_Hz)
```

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- `base` – The I3C peripheral base address.
- `masterConfig` – User provided peripheral configuration. Use `I3C_MasterGetDefaultConfig()` to get a set of defaults that you can override.
- `sourceClock_Hz` – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

```
void I3C_MasterDeinit(I3C_Type *base)
```

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- `base` – The I3C peripheral base address.

```
status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)
```

```
status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)
```

```
status_t I3C_CheckForBusyBus(I3C_Type *base)
```

```
static inline void I3C_MasterEnable(I3C_Type *base, i3c_master_enable_t enable)
```

Set I3C module master mode.

Parameters

- `base` – The I3C peripheral base address.
- `enable` – Enable master mode.

```
void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)
```

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- `slaveConfig` – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- `base` – The I3C peripheral base address.
- `slaveConfig` – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- `slowClock_Hz` – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

```
void I3C_SlaveDeinit(I3C_Type *base)
```

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- `base` – The I3C peripheral base address.

```
static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)
```

Enable/Disable Slave.

Parameters

- `base` – The I3C peripheral base address.
- `isEnabled` – Enable or disable.

```
static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)
```

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_master_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master status flag state.

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag
- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

Attempts to clear other flags has no effect.

See also:

`_i3c_master_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_master_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

```
static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_master_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C master error status flag state.

See also:

`_i3c_master_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_master_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_MasterGetStatusFlags()`.

`i3c_master_state_t` `I3C_MasterGetState(I3C_Type *base)`

Gets the I3C master state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C master state.

`static inline uint32_t` `I3C_SlaveGetStatusFlags(I3C_Type *base)`

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void` `I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

`static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)`

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

`static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)`

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

`i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)`

Gets the I3C slave state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

`status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)`

`static inline void I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Enables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Parameters

- `base` – The I3C peripheral base address.
- `interruptMask` – Bit mask of interrupts to enable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

`static inline void I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)`

Disables the I3C master interrupt requests.

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_master_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_MasterGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_master_flags` enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag

- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C master DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C master transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

```
static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C master receive data register address for DMA transfer.

Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

Parameters

- *base* – The I3C peripheral base address.
- *enableTx* – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- *enableRx* – Enable flag for receive DMA request. Pass true for enable, false for disable.
- *width* – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

Parameters

- *base* – The I3C peripheral base address.
- *width* – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_MasterSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C master FIFOs.

Parameters

- *base* – The I3C peripheral base address.
- *txLvl* – Transmit FIFO watermark level. The `kI3C_MasterTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- *rxLvl* – Receive FIFO watermark level. The `kI3C_MasterRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.

- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C master FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

void I3C_MasterSetBaudRate(I3C_Type *base, const *i3c_baudrate_hz_t* *baudRate_Hz, uint32_t sourceClock_Hz)

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

- `base` – The I3C peripheral base address.
- `baudRate_Hz` – Pointer to structure of requested bus frequency in Hertz.
- `sourceClock_Hz` – I3C functional clock frequency in Hertz.

```
static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)
```

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- *base* – The I3C peripheral base address.

Return values

- *true* – Bus is busy.
- *false* – Bus is idle.

```
status_t I3C_MasterStartWithRxSize(I3C_Type *base, i3c_bus_type_t type, uint8_t address,
                                   i3c_direction_t dir, uint8_t rxSize)
```

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- *rxSize* – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

- *kStatus_Success* – START signal and address were successfully enqueued in the transmit FIFO.
- *kStatus_I3C_Busy* – Another master is currently utilizing the bus.

```
status_t I3C_MasterStart(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)
```

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- *base* – The I3C peripheral base address.
- *type* – The bus type to use in this transaction.
- *address* – 7-bit slave device address, in bits [6:0].
- *dir* – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- `kStatus_Success` – START signal and address were successfully enqueued in the transmit FIFO.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.

`status_t I3C_MasterRepeatedStartWithRxSize(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir, uint8_t rxSize)`

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the `rxSize` = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the `rxSize` configuration.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- `rxSize` – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

`static inline status_t I3C_MasterRepeatedStart(I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)`

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like `I3C_MasterStart()`, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- `base` – The I3C peripheral base address.
- `type` – The bus type to use in this transaction.
- `address` – 7-bit slave device address, in bits [6:0].
- `dir` – Master transfer direction, either `kI3C_Read` or `kI3C_Write`. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

`kStatus_Success` – Repeated START signal and address were successfully enqueued in the transmit FIFO.

status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)

Performs a polling send transfer on the I2C/I3C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns *kStatus_I3C_Nak*.

Parameters

- *base* – The I3C peripheral base address.
- *txBuff* – The pointer to the data to be transferred.
- *txSize* – The length in bytes of the data to be transferred.
- *flags* – Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options.

Return values

- *kStatus_Success* – Data was sent successfully.
- *kStatus_I3C_Busy* – Another master is currently utilizing the bus.
- *kStatus_I3C_Timeout* – The module has stalled too long in a frame.
- *kStatus_I3C_Nak* – The slave device sent a NAK in response to an address.
- *kStatus_I3C_WriteAbort* – The slave device sent a NAK in response to a write.
- *kStatus_I3C_MsgError* – Message SDR/DDR mismatch or read/write message in wrong state.
- *kStatus_I3C_WriteFifoError* – Write to M/SWDATAB register when FIFO full.
- *kStatus_I3C_InvalidReq* – Invalid use of request.

status_t I3C_MasterReceive(I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

Parameters

- *base* – The I3C peripheral base address.
- *rxBuff* – The pointer to the data to be transferred.
- *rxSize* – The length in bytes of the data to be transferred.
- *flags* – Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options.

Return values

- *kStatus_Success* – Data was received successfully.
- *kStatus_I3C_Busy* – Another master is currently utilizing the bus.
- *kStatus_I3C_Timeout* – The module has stalled too long in a frame.
- *kStatus_I3C_Term* – The master terminates slave read.
- *kStatus_I3C_HdrParityError* – Parity error from DDR read.
- *kStatus_I3C_CrcError* – CRC error from DDR read.
- *kStatus_I3C_MsgError* – Message SDR/DDR mismatch or read/write message in wrong state.
- *kStatus_I3C_ReadFifoError* – Read from M/SRDATAB register when FIFO empty.

- `kStatus_I3C_InvalidReq` – Invalid use of request.

`status_t I3C_MasterStop(I3C_Type *base)`

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- `base` – The I3C peripheral base address.

Return values

- `kStatus_Success` – The STOP signal was successfully sent on the bus and the transaction terminated.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`void I3C_MasterEmitRequest(I3C_Type *base, i3c_bus_request_t masterReq)`

I3C master emit request.

Parameters

- `base` – The I3C peripheral base address.
- `masterReq` – I3C master request of type `i3c_bus_request_t`

`static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, i3c_ibi_response_t ibiResponse)`

I3C master emit request.

Parameters

- `base` – The I3C peripheral base address.
- `ibiResponse` – I3C master emit IBI response of type `i3c_ibi_response_t`

`void I3C_MasterRegisterIBI(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)`

I3C master register IBI rule.

Parameters

- `base` – The I3C peripheral base address.
- `ibiRule` – Pointer to ibi rule description of type `i3c_register_ibi_addr_t`

`void I3C_MasterGetIBIRules(I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)`

I3C master get IBI rule.

Parameters

- `base` – The I3C peripheral base address.
- `ibiRule` – Pointer to store the read out ibi rule description.

`i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)`

I3C master get IBI Type.

Parameters

- `base` – The I3C peripheral base address.

Return values

`i3c_ibi_type_t` – Type of `i3c_ibi_type_t`.

```
static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)
```

I3C master get IBI Address.

Parameters

- *base* – The I3C peripheral base address.

Return values

The – 8-bit IBI address.

```
status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t
count, i3c_master_daa_baudrate_t
*daaBaudRate)
```

Performs a DAA in the i3c bus with specified temporary baud rate.

Parameters

- *base* – The I3C peripheral base address.
- *addressList* – The pointer for address list which is used to do DAA.
- *count* – The address count in the address list.
- *daaBaudRate* – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- *kStatus_Success* – The transaction was started successfully.
- *kStatus_I3C_Busy* – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- *kStatus_I3C_SlaveCountExceed* – The I3C slave count has exceed the definition in *I3C_MAX_DEVCNT*.

```
static inline status_t I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t
count)
```

Performs a DAA in the i3c bus.

Parameters

- *base* – The I3C peripheral base address.
- *addressList* – The pointer for address list which is used to do DAA.
- *count* – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- *kStatus_Success* – The transaction was started successfully.
- *kStatus_I3C_Busy* – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- *kStatus_I3C_SlaveCountExceed* – The I3C slave count has exceed the definition in *I3C_MAX_DEVCNT*.

```
i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)
```

Get device information list after DAA process is done.

Parameters

- *base* – The I3C peripheral base address.
- *count* – **[out]** The pointer to store the available device count.

Returns

Pointer to the *i3c_device_info_t* array.

`void I3C_MasterClearDeviceCount(I3C_Type *base)`

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

Parameters

- `base` – The I3C peripheral base address.

`status_t I3C_MasterTransferBlocking(I3C_Type *base, i3c_master_transfer_t *transfer)`

Performs a master polling transfer on the I2C/I3C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- `base` – The I3C peripheral base address.
- `transfer` – Pointer to the transfer structure.

Return values

- `kStatus_Success` – Data was received successfully.
- `kStatus_I3C_Busy` – Another master is currently utilizing the bus.
- `kStatus_I3C_IBIWon` – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- `kStatus_I3C_Timeout` – The module has stalled too long in a frame.
- `kStatus_I3C_Nak` – The slave device sent a NAK in response to an address.
- `kStatus_I3C_WriteAbort` – The slave device sent a NAK in response to a write.
- `kStatus_I3C_Term` – The master terminates slave read.
- `kStatus_I3C_HdrParityError` – Parity error from DDR read.
- `kStatus_I3C_CrcError` – CRC error from DDR read.
- `kStatus_I3C_MsgError` – Message SDR/DDR mismatch or read/write message in wrong state.
- `kStatus_I3C_ReadFifoError` – Read from M/SRDATAB register when FIFO empty.
- `kStatus_I3C_WriteFifoError` – Write to M/SWDATAB register when FIFO full.
- `kStatus_I3C_InvalidReq` – Invalid use of request.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

void I3C_MasterTransferCreateHandle(I3C_Type *base, *i3c_master_handle_t* *handle, const *i3c_master_transfer_callback_t* *callback, void *userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_MasterTransferNonBlocking(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I3C_MasterTransferGetCount(I3C_Type *base, *i3c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` – There is not a non-blocking transaction currently in progress.

`void I3C_MasterTransferAbort(I3C_Type *base, i3c_master_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `intHandle` – Pointer to the I3C master driver handle.

`enum _i3c_master_flags`

I3C master peripheral flags.

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`
- `kI3C_MasterSlave2MasterFlag`

All flags except `kI3C_MasterBetweenFlag` and `kI3C_MasterNackDetectFlag` can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator `kI3C_MasterBetweenFlag`

Between messages/DAAs flag

enumerator `kI3C_MasterNackDetectFlag`

NACK detected flag

enumerator `kI3C_MasterSlaveStartFlag`

Slave request start flag

enumerator kI3C_MasterControlDoneFlag
Master request complete flag

enumerator kI3C_MasterCompleteFlag
Transfer complete flag

enumerator kI3C_MasterRxReadyFlag
Rx data ready in Rx buffer flag

enumerator kI3C_MasterTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_MasterArbitrationWonFlag
Header address won arbitration flag

enumerator kI3C_MasterErrorFlag
Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag
Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum _i3c_master_error_flags
I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag
Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag
Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag
Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag
CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag
Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag
Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag
Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag
The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags
All error flags

enum `_i3c_master_state`

I3C working master state.

Values:

enumerator `kI3C_MasterStateIdle`

Bus stopped.

enumerator `kI3C_MasterStateSlvReq`

Bus stopped but slave holding SDA low.

enumerator `kI3C_MasterStateMsgSdr`

In SDR Message mode from using MWMSG_SDR.

enumerator `kI3C_MasterStateNormAct`

In normal active SDR mode.

enumerator `kI3C_MasterStateDdr`

In DDR Message mode.

enumerator `kI3C_MasterStateDaa`

In ENTDAAs mode.

enumerator `kI3C_MasterStateIbiAck`

Waiting on IBI ACK/NACK decision.

enumerator `kI3C_MasterStateIbiRcv`

Receiving IBI.

enum `_i3c_master_enable`

I3C master enable configuration.

Values:

enumerator `kI3C_MasterOff`

Master off.

enumerator `kI3C_MasterOn`

Master on.

enumerator `kI3C_MasterCapable`

Master capable.

enum `_i3c_master_hkeep`

I3C high keeper configuration.

Values:

enumerator `kI3C_MasterHighKeeperNone`

Use PUR to hold SCL high.

enumerator `kI3C_MasterHighKeeperWiredIn`

Use pin_HK controls.

enumerator `kI3C_MasterPassiveSDA`

Hi-Z for Bus Free and hold SDA.

enumerator `kI3C_MasterPassiveSDASCL`

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum `_i3c_bus_request`

Emits the requested operation when doing in pieces vs. by message.

Values:

enumerator kI3C_RequestNone

No request.

enumerator kI3C_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA

Process DAA.

enumerator kI3C_RequestForceExit

Request to force exit.

enumerator kI3C_RequestAutoIbi

Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type

Bus type with EmitStartAddr.

Values:

enumerator kI3C_TypeI3CSdr

SDR mode of I3C.

enumerator kI3C_TypeI2C

Standard i2c protocol.

enumerator kI3C_TypeI3CDdr

HDR-DDR mode of I3C.

enum _i3c_ibi_response

IBI response.

Values:

enumerator kI3C_IbiRespAck

ACK with no mandatory byte.

enumerator kI3C_IbiRespNack

NACK.

enumerator kI3C_IbiRespAckMandatory

ACK with mandatory byte.

enumerator kI3C_IbiRespManual

Reserved.

enum _i3c_ibi_type

IBI type.

Values:

enumerator kI3C_IbiNormal

In-band interrupt.

enumerator kI3C_IbiHotJoin

slave hot join.

enumerator kI3C_IbiMasterRequest
 slave master ship request.

enum _i3c_ibi_state
 IBI state.

Values:

enumerator kI3C_IbiReady
 In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed
 In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending
 In-band interrupt Ack/Nack pending for decision.

enum _i3c_direction
 Direction of master and slave transfers.
Values:

enumerator kI3C_Write
 Master transmit.

enumerator kI3C_Read
 Master receive.

enum _i3c_tx_trigger_level
 Watermark of TX int/dma trigger level.
Values:

enumerator kI3C_TxTriggerOnEmpty
 Trigger on empty.

enumerator kI3C_TxTriggerUntilOneQuarterOrLess
 Trigger on 1/4 full or less.

enumerator kI3C_TxTriggerUntilOneHalfOrLess
 Trigger on 1/2 full or less.

enumerator kI3C_TxTriggerUntilOneLessThanFull
 Trigger on 1 less than full or less.

enum _i3c_rx_trigger_level
 Watermark of RX int/dma trigger level.
Values:

enumerator kI3C_RxTriggerOnNotEmpty
 Trigger on not empty.

enumerator kI3C_RxTriggerUntilOneQuarterOrMore
 Trigger on 1/4 full or more.

enumerator kI3C_RxTriggerUntilOneHalfOrMore
 Trigger on 1/2 full or more.

enumerator kI3C_RxTriggerUntilThreeQuarterOrMore
 Trigger on 3/4 full or more.

enum _i3c_rx_term_ops

I3C master read termination operations.

Values:

enumerator kI3C_RxTermDisable

Master doesn't terminate read, used for CCC transfer.

enumerator kI3C_RxAutoTerm

Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C_RxTermLastByte

Master terminates read at any time after START, no length limitation.

enum _i3c_start_scl_delay

I3C start SCL delay options.

Values:

enumerator kI3C_NoDelay

No delay.

enumerator kI3C_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum _i3c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the `_i3c_master_transfer::flags` field.

Values:

enumerator kI3C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI3C_TransferNoStopFlag

Don't send a stop condition.

enumerator kI3C_TransferWordsFlag

Transfer in words, else transfer in bytes.

enumerator kI3C_TransferDisableRxTermFlag

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C_TransferRxAutoTermFlag

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in `I3C_MasterReceive`.

enumerator `kI3C_TransferStartWithBroadcastAddr`

Start transfer with 0x7E, then read/write data with device address.

typedef enum `_i3c_master_state` `i3c_master_state_t`

I3C working master state.

typedef enum `_i3c_master_enable` `i3c_master_enable_t`

I3C master enable configuration.

typedef enum `_i3c_master_hkeep` `i3c_master_hkeep_t`

I3C high keeper configuration.

typedef enum `_i3c_bus_request` `i3c_bus_request_t`

Emits the requested operation when doing in pieces vs. by message.

typedef enum `_i3c_bus_type` `i3c_bus_type_t`

Bus type with `EmitStartAddr`.

typedef enum `_i3c_ibi_response` `i3c_ibi_response_t`

IBI response.

typedef enum `_i3c_ibi_type` `i3c_ibi_type_t`

IBI type.

typedef enum `_i3c_ibi_state` `i3c_ibi_state_t`

IBI state.

typedef enum `_i3c_direction` `i3c_direction_t`

Direction of master and slave transfers.

typedef enum `_i3c_tx_trigger_level` `i3c_tx_trigger_level_t`

Watermark of TX int/dma trigger level.

typedef enum `_i3c_rx_trigger_level` `i3c_rx_trigger_level_t`

Watermark of RX int/dma trigger level.

typedef enum `_i3c_rx_term_ops` `i3c_rx_term_ops_t`

I3C master read termination operations.

typedef enum `_i3c_start_scl_delay` `i3c_start_scl_delay_t`

I3C start SCL delay options.

typedef struct `_i3c_register_ibi_addr` `i3c_register_ibi_addr_t`

Structure with setting master IBI rules and slave registry.

typedef struct `_i3c_baudrate` `i3c_baudrate_hz_t`

Structure with I3C baudrate settings.

typedef struct `_i3c_master_daa_baudrate` `i3c_master_daa_baudrate_t`

I3C DAA baud rate configuration.

typedef struct `_i3c_master_config` `i3c_master_config_t`

Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef struct `_i3c_master_transfer` `i3c_master_transfer_t`

typedef struct `_i3c_master_handle` `i3c_master_handle_t`

```
typedef struct _i3c_master_transfer_callback i3c_master_transfer_callback_t
```

i3c master callback functions.

```
typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)
```

Typedef for master interrupt handler.

```
struct _i3c_register_ibi_addr
```

#include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.

Public Members

```
uint8_t address[5]
```

Address array for registry.

```
bool i3cFastStart
```

Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.

```
bool ibiHasPayload
```

Whether the address array has mandatory IBI byte.

```
struct _i3c_baudrate
```

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

```
uint32_t i2cBaud
```

Desired I2C baud rate in Hertz.

```
uint32_t i3cPushPullBaud
```

Desired I3C push-pull baud rate in Hertz.

```
uint32_t i3cOpenDrainBaud
```

Desired I3C open-drain baud rate in Hertz.

```
struct _i3c_master_daa_baudrate
```

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

```
uint32_t sourceClock_Hz
```

FCLK, function clock in Hertz.

```
uint32_t i3cPushPullBaud
```

Desired I3C push-pull baud rate in Hertz.

```
uint32_t i3cOpenDrainBaud
```

Desired I3C open-drain baud rate in Hertz.

```
struct _i3c_master_config
```

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay

I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay

I3C SCL delay after Repeated START.

struct *_i3c_master_transfer_callback*

#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)

Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)

IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Transfer complete callback

struct *_i3c_master_transfer*

#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options. Set to 0 or *kI3C_TransferDefaultFlag* for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i3c_direction_t direction

Either *kI3C_Read* or *kI3C_Write*.

`uint32_t` subaddress
Sub address. Transferred MSB first.

`size_t` subaddressSize
Length of sub address to send in bytes. Maximum size is 4 bytes.

`void *`data
Pointer to data to transfer.

`size_t` dataSize
Number of bytes to transfer.

`i3c_bus_type_t` busType
bus type.

`i3c_ibi_response_t` ibiResponse
ibi response during transfer.

`struct __i3c_master_handle`
`#include <fsl_i3c.h>` Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

`uint8_t` state
Transfer state machine current state.

`uint32_t` remainingBytes
Remaining byte count in current state.

`i3c_rx_term_ops_t` rxTermOps
Read termination operation.

`i3c_master_transfer_t` transfer
Copy of the current transfer info.

`uint8_t` ibiAddress
Slave address which request IBI.

`uint8_t *`ibiBuff
Pointer to IBI buffer to keep ibi bytes.

`size_t` ibiPayloadSize
IBI payload size.

`i3c_ibi_type_t` ibiType
IBI type.

`i3c_master_transfer_callback_t` callback
Callback functions pointer.

`void *`userData
Application data passed to callback.

2.37 I3C Master DMA Driver

```
void I3C_MasterTransferCreateHandleEDMA(I3C_Type *base, i3c_master_edma_handle_t
                                         *handle, const i3c_master_edma_callback_t
                                         *callback, void *userData, edma_handle_t
                                         *rxDmaHandle, edma_handle_t *txDmaHandle)
```

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_MasterTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.
- *rxDmaHandle* – Handle for the DMA receive channel. Created by the user prior to calling this function.
- *txDmaHandle* – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_MasterTransferEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle,
                                i3c_master_transfer_t *transfer)
```

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *transfer* – The pointer to the transfer descriptor.

Return values

- `kStatus_Success` – The transaction was started successfully.
- `kStatus_I3C_Busy` – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

```
status_t I3C_MasterTransferGetCountEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle,
                                         size_t *count)
```

Returns number of bytes transferred so far.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C master driver handle.
- *count* – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- `kStatus_Success` –

- `kStatus_NoTransferInProgress` – There is not a DMA transaction currently in progress.

`void I3C_MasterTransferAbortEDMA(I3C_Type *base, i3c_master_edma_handle_t *handle)`

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to the I3C master driver handle.

`void I3C_MasterTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)`

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- `base` – The I3C peripheral base address.
- `i3cHandle` – Pointer to the I3C master DMA driver handle.

`typedef struct _i3c_master_edma_handle i3c_master_edma_handle_t`

`typedef struct _i3c_master_edma_callback i3c_master_edma_callback_t`
i3c master callback functions.

`struct _i3c_master_edma_callback`
`#include <fsl_i3c_edma.h>` i3c master callback functions.

Public Members

`void (*slave2Master)(I3C_Type *base, void *userData)`

Target asks for controller request.

`void (*ibiCallback)(I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_ibi_type_t ibiType, i3c_ibi_state_t ibiState)`

IBI event callback.

`void (*transferComplete)(I3C_Type *base, i3c_master_edma_handle_t *handle, status_t status, void *userData)`

Transfer complete callback.

`struct _i3c_master_edma_handle`
`#include <fsl_i3c_edma.h>` Driver handle for master EDMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

`I3C_Type *base`
I3C base pointer.

`uint8_t state`
Transfer state machine current state.

`uint32_t transferCount`
Indicates progress of the transfer

`uint8_t subaddressBuffer[4]`
Saving subaddress command.

`uint8_t subaddressCount`
Saving command count.

`i3c_master_transfer_t transfer`
Copy of the current transfer info.

`i3c_master_edma_callback_t callback`
Callback function pointer.

`void *userData`
Application data passed to callback.

`edma_handle_t *rxDmaHandle`
Handle for receive DMA channel.

`edma_handle_t *txDmaHandle`
Handle for transmit DMA channel.

`bool ibiFlag`
IBIWON flag.

`uint8_t ibiAddress`
Slave address which request IBI.

`uint8_t *ibiBuff`
Pointer to IBI buffer to keep ibi bytes.

`size_t ibiPayloadSize`
IBI payload size.

`i3c_ibi_type_t ibiType`
IBI type.

`status_t result`
Transfer result.

2.38 I3C Slave Driver

`void I3C_SlaveGetDefaultConfig(i3c_slave_config_t *slaveConfig)`
Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableSlave = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with `I3C_SlaveInit()`.

Parameters

- slaveConfig – **[out]** User provided configuration structure for default values. Refer to `i3c_slave_config_t`.

```
void I3C_SlaveInit(I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_Hz)
```

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use `I3C_SlaveGetDefaultConfig()` to get a set of defaults that you can override.
- slowClock_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If `FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH` defines as 1, this parameter is useless.

```
void I3C_SlaveDeinit(I3C_Type *base)
```

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- base – The I3C peripheral base address.

```
static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnabled)
```

Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnabled – Enable or disable.

```
static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)
```

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_flags`

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Attempts to clear other flags has no effect.

See also:

`_i3c_slave_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of status flags that are to be cleared. The mask is composed of `_i3c_slave_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetStatusFlags()`.

```
static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)
```

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

`_i3c_slave_error_flags`

Parameters

- `base` – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

```
static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)
```

Clears the I3C slave error status flag state.

See also:

`_i3c_slave_error_flags`.

Parameters

- `base` – The I3C peripheral base address.
- `statusMask` – A bitmask of error status flags that are to be cleared. The mask is composed of `_i3c_slave_error_flags` enumerators OR'd together. You may pass the result of a previous call to `I3C_SlaveGetErrorStatusFlags()`.

```
i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)
```

Gets the I3C slave state.

Parameters

- `base` – The I3C peripheral base address.

Returns

I3C slave activity state, refer `i3c_slave_activity_state_t`.

status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See `_i3c_slave_flags` for the set of constants that should be OR'd together to form the bit mask.

```
static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)
```

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of enabled interrupts.

```
static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)
```

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of `_i3c_slave_flags` enumerators OR'd together to indicate the set of pending interrupts.

```
static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
```

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

```
static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

```
static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)
```

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

```
static inline void I3C_SlaveSetWatermarks(I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
```

Sets the watermarks for I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.

- `txLvl` – Transmit FIFO watermark level. The `kI3C_SlaveTxReadyFlag` flag is set whenever the number of words in the transmit FIFO reaches `txLvl`.
- `rxLvl` – Receive FIFO watermark level. The `kI3C_SlaveRxReadyFlag` flag is set whenever the number of words in the receive FIFO reaches `rxLvl`.
- `flushTx` – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- `flushRx` – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

`static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)`

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- `base` – The I3C peripheral base address.
- `txCount` – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- `rxCount` – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

`status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)`

Performs a polling send transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `txBuff` – The pointer to the data to be transferred.
- `txSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)`

Performs a polling receive transfer on the I3C bus.

Parameters

- `base` – The I3C peripheral base address.
- `rxBuff` – The pointer to the data to be transferred.
- `rxSize` – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

`void I3C_SlaveTransferCreateHandle(I3C_Type *base, i3c_slave_handle_t *handle, i3c_slave_transfer_callback_t callback, void *userData)`

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbort()` API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – **[out]** Pointer to the I3C slave driver handle.
- `callback` – User provided pointer to the asynchronous callback function.
- `userData` – User provided pointer to the application callback data.

`status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)`

Starts accepting slave transfers.

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_SlaveTransferCreateHandle()`. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the `eventMask` parameter to the OR'd combination of `i3c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI3C_SlaveTransmitEvent` and `kI3C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI3C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.
- `eventMask` – Bit mask formed by OR'ing together `i3c_slave_transfer_event_t` enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and `kI3C_SlaveAllEvents` to enable all events.

Return values

- `kStatus_Success` – Slave transfers were successfully started.
- `kStatus_I3C_Busy` – Slave transfers have already been started on this handle.

`status_t I3C_SlaveTransferGetCount(I3C_Type *base, i3c_slave_handle_t *handle, size_t *count)`

Gets the slave transfer status during a non-blocking transfer.

Parameters

- `base` – The I3C peripheral base address.
- `handle` – Pointer to `i2c_slave_handle_t` structure.
- `count` – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- `kStatus_Success` –
- `kStatus_NoTransferInProgress` –

`void I3C_SlaveTransferAbort(I3C_Type *base, i3c_slave_handle_t *handle)`

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

`void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)`

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to struct: `_i3c_slave_handle` structure which stores the transfer state.

`enum _i3c_slave_flags`

I3C slave peripheral flags.

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator `kI3C_SlaveNotStopFlag`

Slave status not stop flag

enumerator `kI3C_SlaveMessageFlag`

Slave status message, indicating slave is listening to the bus traffic or responding

enumerator `kI3C_SlaveRequiredReadFlag`
Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator `kI3C_SlaveRequiredWriteFlag`
Slave status request write, master is doing SDR write to slave, except slave in ENTDAAMode

enumerator `kI3C_SlaveBusDAAFlag`
I3C bus is in ENTDAAMode

enumerator `kI3C_SlaveBusHDRModeFlag`
I3C bus is in HDR mode

enumerator `kI3C_SlaveBusStartFlag`
Start/Re-start event is seen since the bus was last cleared

enumerator `kI3C_SlaveMatchedFlag`
Slave address(dynamic/static) matched since last cleared

enumerator `kI3C_SlaveBusStopFlag`
Stop event is seen since the bus was last cleared

enumerator `kI3C_SlaveRxReadyFlag`
Rx data ready in rx buffer flag

enumerator `kI3C_SlaveTxReadyFlag`
Tx buffer ready for Tx data flag

enumerator `kI3C_SlaveDynamicAddrChangedFlag`
Slave dynamic address has been assigned, re-assigned, or lost

enumerator `kI3C_SlaveReceivedCCCFlag`
Slave received Common command code

enumerator `kI3C_SlaveErrorFlag`
Error occurred flag

enumerator `kI3C_SlaveHDRCommandMatchFlag`
High data rate command match

enumerator `kI3C_SlaveCCCHandledFlag`
Slave received Common command code is handled by I3C module

enumerator `kI3C_SlaveEventSentFlag`
Slave IBI/P2P/MR/HJ event has been sent

enumerator `kI3C_SlaveIbiDisableFlag`
Slave in band interrupt is disabled.

enumerator `kI3C_SlaveMasterRequestDisabledFlag`
Slave master request is disabled.

enumerator `kI3C_SlaveHotJoinDisabledFlag`
Slave Hot-Join is disabled.

enumerator `kI3C_SlaveClearFlags`
All flags which are cleared by the driver upon starting a transfer.

enumerator `kI3C_SlaveAllIrqFlags`

enum _i3c_slave_error_flags

I3C slave error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveErrorOvrerrunFlag

Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C_SlaveErrorUnderrunFlag

Slave internal to-bus buffer/FIFO underrun

enumerator kI3C_SlaveErrorUnderrunNakFlag

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C_SlaveErrorTermFlag

Terminate error from master

enumerator kI3C_SlaveErrorInvalidStartFlag

Slave invalid start flag

enumerator kI3C_SlaveErrorSdrParityFlag

SDR parity error

enumerator kI3C_SlaveErrorHdrParityFlag

HDR parity error

enumerator kI3C_SlaveErrorHdrCRCFlag

HDR-DDR CRC error

enumerator kI3C_SlaveErrorS0S1Flag

S0 or S1 error

enumerator kI3C_SlaveErrorOverreadFlag

Over-read error

enumerator kI3C_SlaveErrorOverwriteFlag

Over-write error

enum _i3c_slave_event

I3C slave.event.

Values:

enumerator kI3C_SlaveEventNormal

Normal mode.

enumerator kI3C_SlaveEventIBI

In band interrupt event.

enumerator kI3C_SlaveEventMasterReq

Master request event.

enumerator kI3C_SlaveEventHotJoinReq

Hot-join event.

enum _i3c_slave_activity_state

I3C slave.activity state.

Values:

enumerator kI3C_SlaveNoLatency

Normal bus operation

enumerator kI3C_SlaveLatency1Ms

1ms of latency.

enumerator kI3C_SlaveLatency100Ms

100ms of latency.

enumerator kI3C_SlaveLatency10S

10s latency.

enum _i3c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I3C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kI3C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kI3C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI3C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveRequiredTransmitEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveStartEvent

A start/repeated start was detected.

enumerator kI3C_SlaveHDRCommandMatchEvent

Slave Match HDR Command.

enumerator kI3C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kI3C_SlaveRequestSentEvent

Slave request event sent.

enumerator kI3C_SlaveReceivedCCCEvent

Slave received CCC event, need to handle by application.

enumerator kI3C_SlaveAllEvents

Bit mask of all available events.

typedef enum _i3c_slave_event i3c_slave_event_t

I3C slave.event.

typedef enum _i3c_slave_activity_state i3c_slave_activity_state_t

I3C slave.activity state.

```
typedef struct _i3c_slave_config i3c_slave_config_t
```

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

```
typedef enum _i3c_slave_transfer_event i3c_slave_transfer_event_t
```

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I3C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

```
typedef struct _i3c_slave_transfer i3c_slave_transfer_t
```

I3C slave transfer structure.

```
typedef struct _i3c_slave_handle i3c_slave_handle_t
```

```
typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C_SlaveSetCallback() function after you have created a handle.

Param base

Base address for the I3C instance on which the event occurred.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
```

Typedef for slave interrupt handler.

```
struct _i3c_slave_config
```

#include <fsl_i3c.h> Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

`uint16_t vendorID`
Device vendor ID(manufacture ID).

`uint32_t partNumber`
Device part number info

`uint8_t dcr`
Device characteristics register information.

`uint8_t bcr`
Bus characteristics register information.

`uint8_t hdrMode`
Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.

`bool nakAllRequest`
Whether to reply NAK to all requests except broadcast CCC.

`bool ignoreS0S1Error`
Whether to ignore S0/S1 error in SDR mode.

`bool offline`
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

`bool matchSlaveStartStop`
Whether to assert start/stop status only the time slave is addressed.

`uint32_t maxWriteLength`
Maximum write length.

`uint32_t maxReadLength`
Maximum read length.

`struct _i3c_slave_transfer`
#include <fsl_i3c.h> I3C slave transfer structure.

Public Members

`uint32_t event`
Reason the callback is being invoked.

`uint8_t *txData`
Transfer buffer

`size_t txDataSize`
Transfer size

`uint8_t *rxData`
Transfer buffer

`size_t rxDataSize`
Transfer size

`status_t completionStatus`
Success or error code describing how the transfer completed. Only applies for `ki3c_slave_completion_event`.

`size_t transferredCount`
Number of bytes actually transferred since start or last repeated start.

struct `_i3c_slave_handle`
#include <fsl_i3c.h> I3C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

i3c_slave_transfer_t transfer
 I3C slave transfer copy.

bool isBusy
 Whether transfer is busy.

bool wasTransmit
 Whether the last transfer was a transmit.

uint32_t eventMask
 Mask of enabled events.

uint32_t transferredCount
 Count of bytes transferred.

i3c_slave_transfer_callback_t callback
 Callback function called at transfer event.

void *userData
 Callback parameter passed to callback.

size_t txFifoSize
 Tx Fifo size

2.39 I3C Slave DMA Driver

```
void I3C_SlaveTransferCreateHandleEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,
                                       i3c_slave_edma_callback_t callback, void *userData,
                                       edma_handle_t *rxDmaHandle, edma_handle_t
                                       *txDmaHandle)
```

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- *base* – The I3C peripheral base address.
- *handle* – Pointer to the I3C slave driver handle.
- *callback* – User provided pointer to the asynchronous callback function.
- *userData* – User provided pointer to the application callback data.
- *rxDmaHandle* – Handle for the DMA receive channel. Created by the user prior to calling this function.

- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

```
status_t I3C_SlaveTransferEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle,  
                              i3c_slave_edma_transfer_t *transfer, uint32_t eventMask)
```

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.
- transfer – The pointer to the transfer descriptor.
- eventMask – Bit mask formed by OR'ing together i3c_slave_transfer_event_t enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.
- kStatus_Fail – The transaction can't be set.

```
void I3C_SlaveTransferAbortEDMA(I3C_Type *base, i3c_slave_edma_handle_t *handle)
```

Abort a slave edma non-blocking transfer in a early time.

Parameters

- base – I3C peripheral base address
- handle – pointer to i3c_slave_edma_handle_t structure

```
void I3C_SlaveTransferEDMAHandleIRQ(I3C_Type *base, void *i3CHandle)
```

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- i3CHandle – Pointer to the I3C slave DMA driver handle.

```
typedef struct _i3c_slave_edma_handle i3c_slave_edma_handle_t
```

```
typedef struct _i3c_slave_edma_transfer i3c_slave_edma_transfer_t
```

I3C slave transfer structure.

```
typedef void (*i3c_slave_edma_callback_t)(I3C_Type *base, i3c_slave_edma_transfer_t *transfer,  
void *userData)
```

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

Param base

Base address for the I3C instance on which the event occurred.

Param handle

Pointer to slave DMA transfer handle.

Param transfer

Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData

Arbitrary pointer-sized value passed from the application.

```
struct __i3c_slave_edma_transfer
```

```
#include <fsl_i3c_edma.h> I3C slave transfer structure.
```

Public Members

```
uint32_t event
```

Reason the callback is being invoked.

```
uint8_t *txData
```

Transfer buffer

```
size_t txDataSize
```

Transfer size

```
uint8_t *rxData
```

Transfer buffer

```
size_t rxDataSize
```

Transfer size

```
status_t completionStatus
```

Success or error code describing how the transfer completed. Only applies for kI3C_SlaveCompletionEvent.

```
struct __i3c_slave_edma_handle
```

```
#include <fsl_i3c_edma.h> I3C slave edma handle structure.
```

Note: The contents of this structure are private and subject to change.

Public Members

```
I3C_Type *base
```

I3C base pointer.

```
i3c_slave_edma_transfer_t transfer
```

I3C slave transfer copy.

```
bool isBusy
```

Whether transfer is busy.

```
bool wasTransmit
```

Whether the last transfer was a transmit.

```
bool isDdrMode
```

Whether this is HDR-DDR transfer.

`uint32_t eventMask`

Mask of enabled events.

`i3c_slave_edma_callback_t callback`

Callback function called at transfer event.

`edma_handle_t *rxDmaHandle`

Handle for receive DMA channel.

`edma_handle_t *txDmaHandle`

Handle for transmit DMA channel.

`void *userData`

Callback parameter passed to callback.

2.40 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_CTIMER2CAPTSEL0`

enumerator `kINPUTMUX_INDEX_CTIMER2CAPTSEL1`

enumerator `kINPUTMUX_INDEX_CTIMER2CAPTSEL2`

enumerator `kINPUTMUX_INDEX_CTIMER2CAPTSEL3`

enumerator `kINPUTMUX_INDEX_CTIMER3CAPTSEL0`

enumerator `kINPUTMUX_INDEX_CTIMER3CAPTSEL1`

enumerator `kINPUTMUX_INDEX_CTIMER3CAPTSEL2`

enumerator `kINPUTMUX_INDEX_CTIMER3CAPTSEL3`

enumerator `kINPUTMUX_INDEX_CTIMER4CAPTSEL0`

enumerator `kINPUTMUX_INDEX_CTIMER4CAPTSEL1`

enumerator `kINPUTMUX_INDEX_CTIMER4CAPTSEL2`

enumerator `kINPUTMUX_INDEX_CTIMER4CAPTSEL3`

enumerator `kINPUTMUX_INDEX_SMARTDMA_TRIGSEL0`

enumerator kINPUTMUX_INDEX_SMARTDMA_TRIGSEL1
enumerator kINPUTMUX_INDEX_SMARTDMA_TRIGSEL2
enumerator kINPUTMUX_INDEX_SMARTDMA_TRIGSEL3
enumerator kINPUTMUX_INDEX_SMARTDMA_TRIGSEL4
enumerator kINPUTMUX_INDEX_SMARTDMA_TRIGSEL5
enumerator kINPUTMUX_INDEX_SMARTDMA_TRIGSEL6
enumerator kINPUTMUX_INDEX_SMARTDMA_TRIGSEL7
enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL0
enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL1
enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL2
enumerator kINPUTMUX_INDEX_ADC0_TRIGSEL3
enumerator kINPUTMUX_INDEX_ADC1_TRIGSEL0
enumerator kINPUTMUX_INDEX_ADC1_TRIGSEL1
enumerator kINPUTMUX_INDEX_ADC1_TRIGSEL2
enumerator kINPUTMUX_INDEX_ADC1_TRIGSEL3
enumerator kINPUTMUX_INDEX_ADC2_TRIGSEL0
enumerator kINPUTMUX_INDEX_ADC2_TRIGSEL1
enumerator kINPUTMUX_INDEX_ADC2_TRIGSEL2
enumerator kINPUTMUX_INDEX_ADC2_TRIGSEL3
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL0
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL1
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL2
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL3
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL4
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL5
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL6
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL7
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL8
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL9
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL10
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL11
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL12
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL13

enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL14
enumerator kINPUTMUX_INDEX_AOI0_TRIGSEL15
enumerator kINPUTMUX_INDEX_TRIG_OUTSEL0
enumerator kINPUTMUX_INDEX_TRIG_OUTSEL1
enumerator kINPUTMUX_INDEX_TRIG_OUTSEL2
enumerator kINPUTMUX_INDEX_TRIG_OUTSEL3
enumerator kINPUTMUX_INDEX_TRIG_OUTSEL4
enumerator kINPUTMUX_INDEX_TRIG_OUTSEL6
enumerator kINPUTMUX_INDEX_TRIG_OUTSEL7
enumerator kINPUTMUX_INDEX_FLEXIO_TRIGSEL0
enumerator kINPUTMUX_INDEX_FLEXIO_TRIGSEL1
enumerator kINPUTMUX_INDEX_FLEXIO_TRIGSEL2
enumerator kINPUTMUX_INDEX_FLEXIO_TRIGSEL3
enumerator kINPUTMUX_INDEX_ENET_TRIGSEL0
enumerator kINPUTMUX_INDEX_ENET_TRIGSEL1

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_Ctinp7ToCtimer0cap
enumerator kINPUTMUX_Ctinp8ToCtimer0cap
enumerator kINPUTMUX_Ctinp9ToCtimer0cap
enumerator kINPUTMUX_Ctinp10ToCtimer0cap
enumerator kINPUTMUX_Ctinp11ToCtimer0cap
enumerator kINPUTMUX_Ctinp12ToCtimer0cap
enumerator kINPUTMUX_Ctinp13ToCtimer0cap
enumerator kINPUTMUX_Ctinp14ToCtimer0cap
enumerator kINPUTMUX_Ctinp15ToCtimer0cap

enumerator kINPUTMUX__Ctinp16ToCtimer0cap
enumerator kINPUTMUX__Ctinp17ToCtimer0cap
enumerator kINPUTMUX__Ctinp18ToCtimer0cap
enumerator kINPUTMUX__Ctinp19ToCtimer0cap
enumerator kINPUTMUX__Aoi0out0ToCtimer0cap
enumerator kINPUTMUX__Aoi0out1ToCtimer0cap
enumerator kINPUTMUX__Aoi0out2ToCtimer0cap
enumerator kINPUTMUX__Aoi0out3ToCtimer0cap
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__Ctimer2mat1ToCtimer0cap
enumerator kINPUTMUX__Ctimer2mat2ToCtimer0cap
enumerator kINPUTMUX__Ctimer2mat3ToCtimer0cap
enumerator kINPUTMUX__Lpi2c0masterendofpacketToCtimer0cap
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToCtimer0cap
enumerator kINPUTMUX__Lpi2c1masterendofpacketToCtimer0cap
enumerator kINPUTMUX__Lpi2c1slaveendofpacketToCtimer0cap
enumerator kINPUTMUX__Lpspi0endofframeToCtimer0cap
enumerator kINPUTMUX__Lpspi0receiveddatawordToCtimer0cap
enumerator kINPUTMUX__Lpspi1endofframeToCtimer0cap
enumerator kINPUTMUX__Lpspi1receiveddatawordToCtimer0cap
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_Lpuart3receiveddatawordToCtimer0cap
enumerator kINPUTMUX_Lpuart3transmitteddatawordToCtimer0cap
enumerator kINPUTMUX_Lpuart3receivelineidleToCtimer0cap
enumerator kINPUTMUX_Lpuart4receiveddatawordToCtimer0cap
enumerator kINPUTMUX_Lpuart4transmitteddatawordToCtimer0cap
enumerator kINPUTMUX_Lpuart4receivelineidleToCtimer0cap
enumerator kINPUTMUX_Adc1tcomp0ToCtimer0cap
enumerator kINPUTMUX_Adc1tcomp1ToCtimer0cap
enumerator kINPUTMUX_Adc1tcomp2ToCtimer0cap
enumerator kINPUTMUX_Adc1tcomp3ToCtimer0cap
enumerator kINPUTMUX_Ctimer3mat1ToCtimer0cap
enumerator kINPUTMUX_Ctimer3mat2ToCtimer0cap
enumerator kINPUTMUX_Ctimer3mat3ToCtimer0cap
enumerator kINPUTMUX_Ctimer4mat1ToCtimer0cap
enumerator kINPUTMUX_Ctimer4mat2ToCtimer0cap
enumerator kINPUTMUX_Ctimer4mat3ToCtimer0cap
enumerator kINPUTMUX_Lpi2c2masterendofpacketToCtimer0cap
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToCtimer0cap
enumerator kINPUTMUX_Lpi2c3masterendofpacketToCtimer0cap
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToCtimer0cap
enumerator kINPUTMUX_Lpuart5receiveddatawordToCtimer0cap
enumerator kINPUTMUX_Lpuart5transmitteddatawordToCtimer0cap
enumerator kINPUTMUX_Lpuart5receivelineidleToCtimer0cap
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__Usb1startofframeToCtimer0cap
enumerator kINPUTMUX__Lpspi2endofframeToCtimer0cap
enumerator kINPUTMUX__Lpspi2receiveddatawordToCtimer0cap
enumerator kINPUTMUX__Lpspi3endofframeToCtimer0cap
enumerator kINPUTMUX__Lpspi3receiveddatawordToCtimer0cap
enumerator kINPUTMUX__Lpspi4endofframeToCtimer0cap
enumerator kINPUTMUX__Lpspi4receiveddatawordToCtimer0cap
enumerator kINPUTMUX__Lpspi5endofframeToCtimer0cap
enumerator kINPUTMUX__Lpspi5receiveddatawordToCtimer0cap
enumerator kINPUTMUX__Ctinp0ToCtimer1cap
enumerator kINPUTMUX__Ctinp1ToCtimer1cap
enumerator kINPUTMUX__Ctinp2ToCtimer1cap
enumerator kINPUTMUX__Ctinp3ToCtimer1cap
enumerator kINPUTMUX__Ctinp4ToCtimer1cap
enumerator kINPUTMUX__Ctinp5ToCtimer1cap
enumerator kINPUTMUX__Ctinp6ToCtimer1cap
enumerator kINPUTMUX__Ctinp7ToCtimer1cap
enumerator kINPUTMUX__Ctinp8ToCtimer1cap
enumerator kINPUTMUX__Ctinp9ToCtimer1cap
enumerator kINPUTMUX__Ctinp10ToCtimer1cap
enumerator kINPUTMUX__Ctinp11ToCtimer1cap
enumerator kINPUTMUX__Ctinp12ToCtimer1cap
enumerator kINPUTMUX__Ctinp13ToCtimer1cap
enumerator kINPUTMUX__Ctinp14ToCtimer1cap
enumerator kINPUTMUX__Ctinp15ToCtimer1cap
enumerator kINPUTMUX__Ctinp16ToCtimer1cap
enumerator kINPUTMUX__Ctinp17ToCtimer1cap
enumerator kINPUTMUX__Ctinp18ToCtimer1cap

enumerator kINPUTMUX__Ctinp19ToCtimer1cap
enumerator kINPUTMUX__Aoi0out0ToCtimer1cap
enumerator kINPUTMUX__Aoi0out1ToCtimer1cap
enumerator kINPUTMUX__Aoi0out2ToCtimer1cap
enumerator kINPUTMUX__Aoi0out3ToCtimer1cap
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__Ctimer2mat1ToCtimer1cap
enumerator kINPUTMUX__Ctimer2mat2ToCtimer1cap
enumerator kINPUTMUX__Ctimer2mat3ToCtimer1cap
enumerator kINPUTMUX__Lpi2c0masterendofpacketToCtimer1cap
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToCtimer1cap
enumerator kINPUTMUX__Lpi2c1masterendofpacketToCtimer1cap
enumerator kINPUTMUX__Lpi2c1slaveendofpacketToCtimer1cap
enumerator kINPUTMUX__Lpspi0endofframeToCtimer1cap
enumerator kINPUTMUX__Lpspi0receiveddatawordToCtimer1cap
enumerator kINPUTMUX__Lpspi1endofframeToCtimer1cap
enumerator kINPUTMUX__Lpspi1receiveddatawordToCtimer1cap
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_Lpuart3receiveddatawordToCtimer1cap
enumerator kINPUTMUX_Lpuart3transmitteddatawordToCtimer1cap
enumerator kINPUTMUX_Lpuart3receivelineidleToCtimer1cap
enumerator kINPUTMUX_Lpuart4receiveddatawordToCtimer1cap
enumerator kINPUTMUX_Lpuart4transmitteddatawordToCtimer1cap
enumerator kINPUTMUX_Lpuart4receivelineidleToCtimer1cap
enumerator kINPUTMUX_Adc1tcomp0ToCtimer1cap
enumerator kINPUTMUX_Adc1tcomp1ToCtimer1cap
enumerator kINPUTMUX_Adc1tcomp2ToCtimer1cap
enumerator kINPUTMUX_Adc1tcomp3ToCtimer1cap
enumerator kINPUTMUX_Ctimer3mat1ToCtimer1cap
enumerator kINPUTMUX_Ctimer3mat2ToCtimer1cap
enumerator kINPUTMUX_Ctimer3mat3ToCtimer1cap
enumerator kINPUTMUX_Ctimer4mat1ToCtimer1cap
enumerator kINPUTMUX_Ctimer4mat2ToCtimer1cap
enumerator kINPUTMUX_Ctimer4mat3ToCtimer1cap
enumerator kINPUTMUX_Lpi2c2masterendofpacketToCtimer1cap
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToCtimer1cap
enumerator kINPUTMUX_Lpi2c3masterendofpacketToCtimer1cap
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToCtimer1cap
enumerator kINPUTMUX_Lpuart5receiveddatawordToCtimer1cap
enumerator kINPUTMUX_Lpuart5transmitteddatawordToCtimer1cap
enumerator kINPUTMUX_Lpuart5receivelineidleToCtimer1cap
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__Usb1startofframeToCtimer1cap
enumerator kINPUTMUX__Lpspi2endofframeToCtimer1cap
enumerator kINPUTMUX__Lpspi2receiveddatawordToCtimer1cap
enumerator kINPUTMUX__Lpspi3endofframeToCtimer1cap
enumerator kINPUTMUX__Lpspi3receiveddatawordToCtimer1cap
enumerator kINPUTMUX__Lpspi4endofframeToCtimer1cap
enumerator kINPUTMUX__Lpspi4receiveddatawordToCtimer1cap
enumerator kINPUTMUX__Lpspi5endofframeToCtimer1cap
enumerator kINPUTMUX__Lpspi5receiveddatawordToCtimer1cap
enumerator kINPUTMUX__Ctinp0ToCtimer2cap
enumerator kINPUTMUX__Ctinp1ToCtimer2cap
enumerator kINPUTMUX__Ctinp2ToCtimer2cap
enumerator kINPUTMUX__Ctinp3ToCtimer2cap
enumerator kINPUTMUX__Ctinp4ToCtimer2cap
enumerator kINPUTMUX__Ctinp5ToCtimer2cap
enumerator kINPUTMUX__Ctinp6ToCtimer2cap
enumerator kINPUTMUX__Ctinp7ToCtimer2cap
enumerator kINPUTMUX__Ctinp8ToCtimer2cap
enumerator kINPUTMUX__Ctinp9ToCtimer2cap
enumerator kINPUTMUX__Ctinp10ToCtimer2cap
enumerator kINPUTMUX__Ctinp11ToCtimer2cap
enumerator kINPUTMUX__Ctinp12ToCtimer2cap
enumerator kINPUTMUX__Ctinp13ToCtimer2cap
enumerator kINPUTMUX__Ctinp14ToCtimer2cap
enumerator kINPUTMUX__Ctinp15ToCtimer2cap
enumerator kINPUTMUX__Ctinp16ToCtimer2cap
enumerator kINPUTMUX__Ctinp17ToCtimer2cap
enumerator kINPUTMUX__Ctinp18ToCtimer2cap
enumerator kINPUTMUX__Ctinp19ToCtimer2cap
enumerator kINPUTMUX__Aoi0out0ToCtimer2cap
enumerator kINPUTMUX__Aoi0out1ToCtimer2cap

enumerator kINPUTMUX__Aoi0out2ToCtimer2cap
enumerator kINPUTMUX__Aoi0out3ToCtimer2cap
enumerator kINPUTMUX__Adc0tcomp0ToCtimer2cap
enumerator kINPUTMUX__Adc0tcomp1ToCtimer2cap
enumerator kINPUTMUX__Adc0tcomp2ToCtimer2cap
enumerator kINPUTMUX__Adc0tcomp3ToCtimer2cap
enumerator kINPUTMUX__Cmp0outToCtimer2cap
enumerator kINPUTMUX__Ctimer0mat1ToCtimer2cap
enumerator kINPUTMUX__Ctimer0mat2ToCtimer2cap
enumerator kINPUTMUX__Ctimer0mat3ToCtimer2cap
enumerator kINPUTMUX__Ctimer1mat1ToCtimer2cap
enumerator kINPUTMUX__Ctimer1mat2ToCtimer2cap
enumerator kINPUTMUX__Ctimer1mat3ToCtimer2cap
enumerator kINPUTMUX__Lpi2c0masterendofpacketToCtimer2cap
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToCtimer2cap
enumerator kINPUTMUX__Lpi2c1masterendofpacketToCtimer2cap
enumerator kINPUTMUX__Lpi2c1slaveendofpacketToCtimer2cap
enumerator kINPUTMUX__Lpspi0endofframeToCtimer2cap
enumerator kINPUTMUX__Lpspi0receiveddatawordToCtimer2cap
enumerator kINPUTMUX__Lpspi1endofframeToCtimer2cap
enumerator kINPUTMUX__Lpspi1receiveddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart0receiveddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart0transmitteddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart0receivelineidleToCtimer2cap
enumerator kINPUTMUX__Lpuart1receiveddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart1transmitteddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart1receivelineidleToCtimer2cap
enumerator kINPUTMUX__Lpuart2receiveddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart2transmitteddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart2receivelineidleToCtimer2cap
enumerator kINPUTMUX__Lpuart3receiveddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart3transmitteddatawordToCtimer2cap
enumerator kINPUTMUX__Lpuart3receivelineidleToCtimer2cap

enumerator kINPUTMUX_Lpuart4receiveddatawordToCtimer2cap
enumerator kINPUTMUX_Lpuart4transmitteddatawordToCtimer2cap
enumerator kINPUTMUX_Lpuart4receivelineidleToCtimer2cap
enumerator kINPUTMUX_Adc1tcomp0ToCtimer2cap
enumerator kINPUTMUX_Adc1tcomp1ToCtimer2cap
enumerator kINPUTMUX_Adc1tcomp2ToCtimer2cap
enumerator kINPUTMUX_Adc1tcomp3ToCtimer2cap
enumerator kINPUTMUX_Ctimer3mat1ToCtimer2cap
enumerator kINPUTMUX_Ctimer3mat2ToCtimer2cap
enumerator kINPUTMUX_Ctimer3mat3ToCtimer2cap
enumerator kINPUTMUX_Ctimer4mat1ToCtimer2cap
enumerator kINPUTMUX_Ctimer4mat2ToCtimer2cap
enumerator kINPUTMUX_Ctimer4mat3ToCtimer2cap
enumerator kINPUTMUX_Lpi2c2masterendofpacketToCtimer2cap
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToCtimer2cap
enumerator kINPUTMUX_Lpi2c3masterendofpacketToCtimer2cap
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToCtimer2cap
enumerator kINPUTMUX_Lpuart5receiveddatawordToCtimer2cap
enumerator kINPUTMUX_Lpuart5transmitteddatawordToCtimer2cap
enumerator kINPUTMUX_Lpuart5receivelineidleToCtimer2cap
enumerator kINPUTMUX_Trigin0ToCtimer2cap
enumerator kINPUTMUX_Trigin1ToCtimer2cap
enumerator kINPUTMUX_Trigin2ToCtimer2cap
enumerator kINPUTMUX_Trigin3ToCtimer2cap
enumerator kINPUTMUX_Trigin4ToCtimer2cap
enumerator kINPUTMUX_Trigin5ToCtimer2cap
enumerator kINPUTMUX_Trigin6ToCtimer2cap
enumerator kINPUTMUX_Trigin7ToCtimer2cap
enumerator kINPUTMUX_Trigin8ToCtimer2cap
enumerator kINPUTMUX_Trigin9ToCtimer2cap
enumerator kINPUTMUX_Trigin10ToCtimer2cap
enumerator kINPUTMUX_Trigin11ToCtimer2cap
enumerator kINPUTMUX_Usb1startofframeToCtimer2cap

enumerator kINPUTMUX_Lpspi2endofframeToCtimer2cap
enumerator kINPUTMUX_Lpspi2receiveddatawordToCtimer2cap
enumerator kINPUTMUX_Lpspi3endofframeToCtimer2cap
enumerator kINPUTMUX_Lpspi3receiveddatawordToCtimer2cap
enumerator kINPUTMUX_Lpspi4endofframeToCtimer2cap
enumerator kINPUTMUX_Lpspi4receiveddatawordToCtimer2cap
enumerator kINPUTMUX_Lpspi5endofframeToCtimer2cap
enumerator kINPUTMUX_Lpspi5receiveddatawordToCtimer2cap
enumerator kINPUTMUX_Ctinp0ToCtimer3cap
enumerator kINPUTMUX_Ctinp1ToCtimer3cap
enumerator kINPUTMUX_Ctinp2ToCtimer3cap
enumerator kINPUTMUX_Ctinp3ToCtimer3cap
enumerator kINPUTMUX_Ctinp4ToCtimer3cap
enumerator kINPUTMUX_Ctinp5ToCtimer3cap
enumerator kINPUTMUX_Ctinp6ToCtimer3cap
enumerator kINPUTMUX_Ctinp7ToCtimer3cap
enumerator kINPUTMUX_Ctinp8ToCtimer3cap
enumerator kINPUTMUX_Ctinp9ToCtimer3cap
enumerator kINPUTMUX_Ctinp10ToCtimer3cap
enumerator kINPUTMUX_Ctinp11ToCtimer3cap
enumerator kINPUTMUX_Ctinp12ToCtimer3cap
enumerator kINPUTMUX_Ctinp13ToCtimer3cap
enumerator kINPUTMUX_Ctinp14ToCtimer3cap
enumerator kINPUTMUX_Ctinp15ToCtimer3cap
enumerator kINPUTMUX_Ctinp16ToCtimer3cap
enumerator kINPUTMUX_Ctinp17ToCtimer3cap
enumerator kINPUTMUX_Ctinp18ToCtimer3cap
enumerator kINPUTMUX_Ctinp19ToCtimer3cap
enumerator kINPUTMUX_Aoi0out0ToCtimer3cap
enumerator kINPUTMUX_Aoi0out1ToCtimer3cap
enumerator kINPUTMUX_Aoi0out2ToCtimer3cap
enumerator kINPUTMUX_Aoi0out3ToCtimer3cap
enumerator kINPUTMUX_Adc0tcomp0ToCtimer3cap

enumerator kINPUTMUX__Adc0tcomp1ToCtimer3cap
enumerator kINPUTMUX__Adc0tcomp2ToCtimer3cap
enumerator kINPUTMUX__Adc0tcomp3ToCtimer3cap
enumerator kINPUTMUX__Cmp0outToCtimer3cap
enumerator kINPUTMUX__Ctimer0mat1ToCtimer3cap
enumerator kINPUTMUX__Ctimer0mat2ToCtimer3cap
enumerator kINPUTMUX__Ctimer0mat3ToCtimer3cap
enumerator kINPUTMUX__Ctimer1mat1ToCtimer3cap
enumerator kINPUTMUX__Ctimer1mat2ToCtimer3cap
enumerator kINPUTMUX__Ctimer1mat3ToCtimer3cap
enumerator kINPUTMUX__Lpi2c0masterendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpi2c1masterendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpi2c1slaveendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpspi0endofframeToCtimer3cap
enumerator kINPUTMUX__Lpspi0receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpspi1endofframeToCtimer3cap
enumerator kINPUTMUX__Lpspi1receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart0receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart0transmitteddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart0receivelineidleToCtimer3cap
enumerator kINPUTMUX__Lpuart1receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart1transmitteddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart1receivelineidleToCtimer3cap
enumerator kINPUTMUX__Lpuart2receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart2transmitteddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart2receivelineidleToCtimer3cap
enumerator kINPUTMUX__Lpuart3receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart3transmitteddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart3receivelineidleToCtimer3cap
enumerator kINPUTMUX__Lpuart4receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart4transmitteddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart4receivelineidleToCtimer3cap

enumerator kINPUTMUX__Adc1tcomp0ToCtimer3cap
enumerator kINPUTMUX__Adc1tcomp1ToCtimer3cap
enumerator kINPUTMUX__Adc1tcomp2ToCtimer3cap
enumerator kINPUTMUX__Adc1tcomp3ToCtimer3cap
enumerator kINPUTMUX__Ctimer2mat1ToCtimer3cap
enumerator kINPUTMUX__Ctimer2mat2ToCtimer3cap
enumerator kINPUTMUX__Ctimer2mat3ToCtimer3cap
enumerator kINPUTMUX__Ctimer4mat1ToCtimer3cap
enumerator kINPUTMUX__Ctimer4mat2ToCtimer3cap
enumerator kINPUTMUX__Ctimer4mat3ToCtimer3cap
enumerator kINPUTMUX__Lpi2c2masterendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpi2c2slaveendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpi2c3masterendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpi2c3slaveendofpacketToCtimer3cap
enumerator kINPUTMUX__Lpuart5receiveddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart5transmitteddatawordToCtimer3cap
enumerator kINPUTMUX__Lpuart5receivelineidleToCtimer3cap
enumerator kINPUTMUX__Tmprout0ToCtimer3cap
enumerator kINPUTMUX__Tmprout1ToCtimer3cap
enumerator kINPUTMUX__Trigin0ToCtimer3cap
enumerator kINPUTMUX__Trigin1ToCtimer3cap
enumerator kINPUTMUX__Trigin2ToCtimer3cap
enumerator kINPUTMUX__Trigin3ToCtimer3cap
enumerator kINPUTMUX__Trigin4ToCtimer3cap
enumerator kINPUTMUX__Trigin5ToCtimer3cap
enumerator kINPUTMUX__Trigin6ToCtimer3cap
enumerator kINPUTMUX__Trigin7ToCtimer3cap
enumerator kINPUTMUX__Trigin8ToCtimer3cap
enumerator kINPUTMUX__Trigin9ToCtimer3cap
enumerator kINPUTMUX__Trigin10ToCtimer3cap
enumerator kINPUTMUX__Trigin11ToCtimer3cap
enumerator kINPUTMUX__Usb1startofframeToCtimer3cap
enumerator kINPUTMUX__Lpspi2endofframeToCtimer3cap

enumerator kINPUTMUX_Lpspi2receiveddatawordToCtimer3cap
enumerator kINPUTMUX_Lpspi3endofframeToCtimer3cap
enumerator kINPUTMUX_Lpspi3receiveddatawordToCtimer3cap
enumerator kINPUTMUX_Lpspi4endofframeToCtimer3cap
enumerator kINPUTMUX_Lpspi4receiveddatawordToCtimer3cap
enumerator kINPUTMUX_Lpspi5endofframeToCtimer3cap
enumerator kINPUTMUX_Lpspi5receiveddatawordToCtimer3cap
enumerator kINPUTMUX_Ctinp0ToCtimer4cap
enumerator kINPUTMUX_Ctinp1ToCtimer4cap
enumerator kINPUTMUX_Ctinp2ToCtimer4cap
enumerator kINPUTMUX_Ctinp3ToCtimer4cap
enumerator kINPUTMUX_Ctinp4ToCtimer4cap
enumerator kINPUTMUX_Ctinp5ToCtimer4cap
enumerator kINPUTMUX_Ctinp6ToCtimer4cap
enumerator kINPUTMUX_Ctinp7ToCtimer4cap
enumerator kINPUTMUX_Ctinp8ToCtimer4cap
enumerator kINPUTMUX_Ctinp9ToCtimer4cap
enumerator kINPUTMUX_Ctinp10ToCtimer4cap
enumerator kINPUTMUX_Ctinp11ToCtimer4cap
enumerator kINPUTMUX_Ctinp12ToCtimer4cap
enumerator kINPUTMUX_Ctinp13ToCtimer4cap
enumerator kINPUTMUX_Ctinp14ToCtimer4cap
enumerator kINPUTMUX_Ctinp15ToCtimer4cap
enumerator kINPUTMUX_Ctinp16ToCtimer4cap
enumerator kINPUTMUX_Ctinp17ToCtimer4cap
enumerator kINPUTMUX_Ctinp18ToCtimer4cap
enumerator kINPUTMUX_Ctinp19ToCtimer4cap
enumerator kINPUTMUX_Aoi0out0ToCtimer4cap
enumerator kINPUTMUX_Aoi0out1ToCtimer4cap
enumerator kINPUTMUX_Aoi0out2ToCtimer4cap
enumerator kINPUTMUX_Aoi0out3ToCtimer4cap
enumerator kINPUTMUX_Adc0tcomp0ToCtimer4cap
enumerator kINPUTMUX_Adc0tcomp1ToCtimer4cap

enumerator kINPUTMUX__Adc0tcomp2ToCtimer4cap
enumerator kINPUTMUX__Adc0tcomp3ToCtimer4cap
enumerator kINPUTMUX__Cmp0outToCtimer4cap
enumerator kINPUTMUX__Ctimer0mat1ToCtimer4cap
enumerator kINPUTMUX__Ctimer0mat2ToCtimer4cap
enumerator kINPUTMUX__Ctimer0mat3ToCtimer4cap
enumerator kINPUTMUX__Ctimer1mat1ToCtimer4cap
enumerator kINPUTMUX__Ctimer1mat2ToCtimer4cap
enumerator kINPUTMUX__Ctimer1mat3ToCtimer4cap
enumerator kINPUTMUX__Lpi2c0masterendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpi2c1masterendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpi2c1slaveendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpspi0endofframeToCtimer4cap
enumerator kINPUTMUX__Lpspi0receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpspi1endofframeToCtimer4cap
enumerator kINPUTMUX__Lpspi1receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart0receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart0transmitteddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart0receivelineidleToCtimer4cap
enumerator kINPUTMUX__Lpuart1receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart1transmitteddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart1receivelineidleToCtimer4cap
enumerator kINPUTMUX__Lpuart2receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart2transmitteddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart2receivelineidleToCtimer4cap
enumerator kINPUTMUX__Lpuart3receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart3transmitteddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart3receivelineidleToCtimer4cap
enumerator kINPUTMUX__Lpuart4receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart4transmitteddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart4receivelineidleToCtimer4cap
enumerator kINPUTMUX__Adc1tcomp0ToCtimer4cap

enumerator kINPUTMUX__Adc1tcomp1ToCtimer4cap
enumerator kINPUTMUX__Adc1tcomp2ToCtimer4cap
enumerator kINPUTMUX__Adc1tcomp3ToCtimer4cap
enumerator kINPUTMUX__Ctimer2mat1ToCtimer4cap
enumerator kINPUTMUX__Ctimer2mat2ToCtimer4cap
enumerator kINPUTMUX__Ctimer2mat3ToCtimer4cap
enumerator kINPUTMUX__Ctimer3mat1ToCtimer4cap
enumerator kINPUTMUX__Ctimer3mat2ToCtimer4cap
enumerator kINPUTMUX__Ctimer3mat3ToCtimer4cap
enumerator kINPUTMUX__Lpi2c2masterendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpi2c2slaveendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpi2c3masterendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpi2c3slaveendofpacketToCtimer4cap
enumerator kINPUTMUX__Lpuart5receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart5transmitteddatawordToCtimer4cap
enumerator kINPUTMUX__Lpuart5receivelineidleToCtimer4cap
enumerator kINPUTMUX__Trigin0ToCtimer4cap
enumerator kINPUTMUX__Trigin1ToCtimer4cap
enumerator kINPUTMUX__Trigin2ToCtimer4cap
enumerator kINPUTMUX__Trigin3ToCtimer4cap
enumerator kINPUTMUX__Trigin4ToCtimer4cap
enumerator kINPUTMUX__Trigin5ToCtimer4cap
enumerator kINPUTMUX__Trigin6ToCtimer4cap
enumerator kINPUTMUX__Trigin7ToCtimer4cap
enumerator kINPUTMUX__Trigin8ToCtimer4cap
enumerator kINPUTMUX__Trigin9ToCtimer4cap
enumerator kINPUTMUX__Trigin10ToCtimer4cap
enumerator kINPUTMUX__Trigin11ToCtimer4cap
enumerator kINPUTMUX__Usb1startofframeToCtimer4cap
enumerator kINPUTMUX__Lpspi2endofframeToCtimer4cap
enumerator kINPUTMUX__Lpspi2receiveddatawordToCtimer4cap
enumerator kINPUTMUX__Lpspi3endofframeToCtimer4cap
enumerator kINPUTMUX__Lpspi3receiveddatawordToCtimer4cap

enumerator kINPUTMUX_Lpspi4endofframeToCtimer4cap
enumerator kINPUTMUX_Lpspi4receiveddatawordToCtimer4cap
enumerator kINPUTMUX_Lpspi5endofframeToCtimer4cap
enumerator kINPUTMUX_Lpspi5receiveddatawordToCtimer4cap
enumerator kINPUTMUX_Ctinp0ToTimer0trig
enumerator kINPUTMUX_Ctinp1ToTimer0trig
enumerator kINPUTMUX_Ctinp2ToTimer0trig
enumerator kINPUTMUX_Ctinp3ToTimer0trig
enumerator kINPUTMUX_Ctinp4ToTimer0trig
enumerator kINPUTMUX_Ctinp5ToTimer0trig
enumerator kINPUTMUX_Ctinp6ToTimer0trig
enumerator kINPUTMUX_Ctinp7ToTimer0trig
enumerator kINPUTMUX_Ctinp8ToTimer0trig
enumerator kINPUTMUX_Ctinp9ToTimer0trig
enumerator kINPUTMUX_Ctinp10ToTimer0trig
enumerator kINPUTMUX_Ctinp11ToTimer0trig
enumerator kINPUTMUX_Ctinp12ToTimer0trig
enumerator kINPUTMUX_Ctinp13ToTimer0trig
enumerator kINPUTMUX_Ctinp14ToTimer0trig
enumerator kINPUTMUX_Ctinp15ToTimer0trig
enumerator kINPUTMUX_Ctinp16ToTimer0trig
enumerator kINPUTMUX_Ctinp17ToTimer0trig
enumerator kINPUTMUX_Ctinp18ToTimer0trig
enumerator kINPUTMUX_Ctinp19ToTimer0trig
enumerator kINPUTMUX_Aoi0out0ToTimer0trig
enumerator kINPUTMUX_Aoi0out1ToTimer0trig
enumerator kINPUTMUX_Aoi0out2ToTimer0trig
enumerator kINPUTMUX_Aoi0out3ToTimer0trig
enumerator kINPUTMUX_Adc0tcomp0ToTimer0trig
enumerator kINPUTMUX_Adc0tcomp1ToTimer0trig
enumerator kINPUTMUX_Adc0tcomp2ToTimer0trig
enumerator kINPUTMUX_Adc0tcomp3ToTimer0trig
enumerator kINPUTMUX_Cmp0outToTimer0trig

enumerator kINPUTMUX_Ctimer1mat1ToTimer0trig
enumerator kINPUTMUX_Ctimer1mat2ToTimer0trig
enumerator kINPUTMUX_Ctimer1mat3ToTimer0trig
enumerator kINPUTMUX_Ctimer2mat1ToTimer0trig
enumerator kINPUTMUX_Ctimer2mat2ToTimer0trig
enumerator kINPUTMUX_Ctimer2mat3ToTimer0trig
enumerator kINPUTMUX_Lpi2c0masterendofpacketToTimer0trig
enumerator kINPUTMUX_Lpi2c0slaveendofpacketToTimer0trig
enumerator kINPUTMUX_Lpi2c1masterendofpacketToTimer0trig
enumerator kINPUTMUX_Lpi2c1slaveendofpacketToTimer0trig
enumerator kINPUTMUX_Lpspi0endofframeToTimer0trig
enumerator kINPUTMUX_Lpspi0receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpspi1endofframeToTimer0trig
enumerator kINPUTMUX_Lpspi1receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart0receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart0transmitteddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart0receivelineidleToTimer0trig
enumerator kINPUTMUX_Lpuart1receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart1transmitteddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart1receivelineidleToTimer0trig
enumerator kINPUTMUX_Lpuart2receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart2transmitteddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart2receivelineidleToTimer0trig
enumerator kINPUTMUX_Lpuart3receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart3transmitteddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart3receivelineidleToTimer0trig
enumerator kINPUTMUX_Lpuart4receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart4transmitteddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart4receivelineidleToTimer0trig
enumerator kINPUTMUX_Adc1tcomp0ToTimer0trig
enumerator kINPUTMUX_Adc1tcomp1ToTimer0trig
enumerator kINPUTMUX_Adc1tcomp2ToTimer0trig
enumerator kINPUTMUX_Adc1tcomp3ToTimer0trig

enumerator kINPUTMUX_Ctimer3mat1ToTimer0trig
enumerator kINPUTMUX_Ctimer3mat2ToTimer0trig
enumerator kINPUTMUX_Ctimer3mat3ToTimer0trig
enumerator kINPUTMUX_Ctimer4mat1ToTimer0trig
enumerator kINPUTMUX_Ctimer4mat2ToTimer0trig
enumerator kINPUTMUX_Ctimer4mat3ToTimer0trig
enumerator kINPUTMUX_Lpi2c2masterendofpacketToTimer0trig
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToTimer0trig
enumerator kINPUTMUX_Lpi2c3masterendofpacketToTimer0trig
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToTimer0trig
enumerator kINPUTMUX_Lpuart5receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart5transmitteddatawordToTimer0trig
enumerator kINPUTMUX_Lpuart5receivelineidleToTimer0trig
enumerator kINPUTMUX_Trigin0ToTimer0trig
enumerator kINPUTMUX_Trigin1ToTimer0trig
enumerator kINPUTMUX_Trigin2ToTimer0trig
enumerator kINPUTMUX_Trigin3ToTimer0trig
enumerator kINPUTMUX_Trigin4ToTimer0trig
enumerator kINPUTMUX_Trigin5ToTimer0trig
enumerator kINPUTMUX_Trigin6ToTimer0trig
enumerator kINPUTMUX_Trigin7ToTimer0trig
enumerator kINPUTMUX_Trigin8ToTimer0trig
enumerator kINPUTMUX_Trigin9ToTimer0trig
enumerator kINPUTMUX_Trigin10ToTimer0trig
enumerator kINPUTMUX_Trigin11ToTimer0trig
enumerator kINPUTMUX_Usb1startofframeToTimer0trig
enumerator kINPUTMUX_Lpspi2endofframeToTimer0trig
enumerator kINPUTMUX_Lpspi2receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpspi3endofframeToTimer0trig
enumerator kINPUTMUX_Lpspi3receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpspi4endofframeToTimer0trig
enumerator kINPUTMUX_Lpspi4receiveddatawordToTimer0trig
enumerator kINPUTMUX_Lpspi5endofframeToTimer0trig

enumerator kINPUTMUX_Lpspi5receiveddatawordToTimer0trig
enumerator kINPUTMUX_Ctinp0ToTimer1trig
enumerator kINPUTMUX_Ctinp1ToTimer1trig
enumerator kINPUTMUX_Ctinp2ToTimer1trig
enumerator kINPUTMUX_Ctinp3ToTimer1trig
enumerator kINPUTMUX_Ctinp4ToTimer1trig
enumerator kINPUTMUX_Ctinp5ToTimer1trig
enumerator kINPUTMUX_Ctinp6ToTimer1trig
enumerator kINPUTMUX_Ctinp7ToTimer1trig
enumerator kINPUTMUX_Ctinp8ToTimer1trig
enumerator kINPUTMUX_Ctinp9ToTimer1trig
enumerator kINPUTMUX_Ctinp10ToTimer1trig
enumerator kINPUTMUX_Ctinp11ToTimer1trig
enumerator kINPUTMUX_Ctinp12ToTimer1trig
enumerator kINPUTMUX_Ctinp13ToTimer1trig
enumerator kINPUTMUX_Ctinp14ToTimer1trig
enumerator kINPUTMUX_Ctinp15ToTimer1trig
enumerator kINPUTMUX_Ctinp16ToTimer1trig
enumerator kINPUTMUX_Ctinp17ToTimer1trig
enumerator kINPUTMUX_Ctinp18ToTimer1trig
enumerator kINPUTMUX_Ctinp19ToTimer1trig
enumerator kINPUTMUX_Aoi0out0ToTimer1trig
enumerator kINPUTMUX_Aoi0out1ToTimer1trig
enumerator kINPUTMUX_Aoi0out2ToTimer1trig
enumerator kINPUTMUX_Aoi0out3ToTimer1trig
enumerator kINPUTMUX_Adc0tcomp0ToTimer1trig
enumerator kINPUTMUX_Adc0tcomp1ToTimer1trig
enumerator kINPUTMUX_Adc0tcomp2ToTimer1trig
enumerator kINPUTMUX_Adc0tcomp3ToTimer1trig
enumerator kINPUTMUX_Cmp0outToTimer1trig
enumerator kINPUTMUX_Ctimer0mat1ToTimer1trig
enumerator kINPUTMUX_Ctimer0mat2ToTimer1trig
enumerator kINPUTMUX_Ctimer0mat3ToTimer1trig

enumerator kINPUTMUX_Ctimer2mat1ToTimer1trig
enumerator kINPUTMUX_Ctimer2mat2ToTimer1trig
enumerator kINPUTMUX_Ctimer2mat3ToTimer1trig
enumerator kINPUTMUX_Lpi2c0masterendofpacketToTimer1trig
enumerator kINPUTMUX_Lpi2c0slaveendofpacketToTimer1trig
enumerator kINPUTMUX_Lpi2c1masterendofpacketToTimer1trig
enumerator kINPUTMUX_Lpi2c1slaveendofpacketToTimer1trig
enumerator kINPUTMUX_Lpspi0endofframeToTimer1trig
enumerator kINPUTMUX_Lpspi0receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpspi1endofframeToTimer1trig
enumerator kINPUTMUX_Lpspi1receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart0receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart0transmitteddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart0receivelineidleToTimer1trig
enumerator kINPUTMUX_Lpuart1receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart1transmitteddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart1receivelineidleToTimer1trig
enumerator kINPUTMUX_Lpuart2receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart2transmitteddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart2receivelineidleToTimer1trig
enumerator kINPUTMUX_Lpuart3receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart3transmitteddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart3receivelineidleToTimer1trig
enumerator kINPUTMUX_Lpuart4receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart4transmitteddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart4receivelineidleToTimer1trig
enumerator kINPUTMUX_Adc1tcomp0ToTimer1trig
enumerator kINPUTMUX_Adc1tcomp1ToTimer1trig
enumerator kINPUTMUX_Adc1tcomp2ToTimer1trig
enumerator kINPUTMUX_Adc1tcomp3ToTimer1trig
enumerator kINPUTMUX_Ctimer3mat1ToTimer1trig
enumerator kINPUTMUX_Ctimer3mat2ToTimer1trig
enumerator kINPUTMUX_Ctimer3mat3ToTimer1trig

enumerator kINPUTMUX_Ctimer4mat1ToTimer1trig
enumerator kINPUTMUX_Ctimer4mat2ToTimer1trig
enumerator kINPUTMUX_Ctimer4mat3ToTimer1trig
enumerator kINPUTMUX_Lpi2c2masterendofpacketToTimer1trig
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToTimer1trig
enumerator kINPUTMUX_Lpi2c3masterendofpacketToTimer1trig
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToTimer1trig
enumerator kINPUTMUX_Lpuart5receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart5transmitteddatawordToTimer1trig
enumerator kINPUTMUX_Lpuart5receivelineidleToTimer1trig
enumerator kINPUTMUX_Trigin0ToTimer1trig
enumerator kINPUTMUX_Trigin1ToTimer1trig
enumerator kINPUTMUX_Trigin2ToTimer1trig
enumerator kINPUTMUX_Trigin3ToTimer1trig
enumerator kINPUTMUX_Trigin4ToTimer1trig
enumerator kINPUTMUX_Trigin5ToTimer1trig
enumerator kINPUTMUX_Trigin6ToTimer1trig
enumerator kINPUTMUX_Trigin7ToTimer1trig
enumerator kINPUTMUX_Trigin8ToTimer1trig
enumerator kINPUTMUX_Trigin9ToTimer1trig
enumerator kINPUTMUX_Trigin10ToTimer1trig
enumerator kINPUTMUX_Trigin11ToTimer1trig
enumerator kINPUTMUX_Usb1startofframeToTimer1trig
enumerator kINPUTMUX_Lpspi2endofframeToTimer1trig
enumerator kINPUTMUX_Lpspi2receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpspi3endofframeToTimer1trig
enumerator kINPUTMUX_Lpspi3receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpspi4endofframeToTimer1trig
enumerator kINPUTMUX_Lpspi4receiveddatawordToTimer1trig
enumerator kINPUTMUX_Lpspi5endofframeToTimer1trig
enumerator kINPUTMUX_Lpspi5receiveddatawordToTimer1trig
enumerator kINPUTMUX_Ctinp0ToTimer2trig
enumerator kINPUTMUX_Ctinp1ToTimer2trig

enumerator kINPUTMUX__Ctinp2ToTimer2trig
enumerator kINPUTMUX__Ctinp3ToTimer2trig
enumerator kINPUTMUX__Ctinp4ToTimer2trig
enumerator kINPUTMUX__Ctinp5ToTimer2trig
enumerator kINPUTMUX__Ctinp6ToTimer2trig
enumerator kINPUTMUX__Ctinp7ToTimer2trig
enumerator kINPUTMUX__Ctinp8ToTimer2trig
enumerator kINPUTMUX__Ctinp9ToTimer2trig
enumerator kINPUTMUX__Ctinp10ToTimer2trig
enumerator kINPUTMUX__Ctinp11ToTimer2trig
enumerator kINPUTMUX__Ctinp12ToTimer2trig
enumerator kINPUTMUX__Ctinp13ToTimer2trig
enumerator kINPUTMUX__Ctinp14ToTimer2trig
enumerator kINPUTMUX__Ctinp15ToTimer2trig
enumerator kINPUTMUX__Ctinp16ToTimer2trig
enumerator kINPUTMUX__Ctinp17ToTimer2trig
enumerator kINPUTMUX__Ctinp18ToTimer2trig
enumerator kINPUTMUX__Ctinp19ToTimer2trig
enumerator kINPUTMUX__Aoi0out0ToTimer2trig
enumerator kINPUTMUX__Aoi0out1ToTimer2trig
enumerator kINPUTMUX__Aoi0out2ToTimer2trig
enumerator kINPUTMUX__Aoi0out3ToTimer2trig
enumerator kINPUTMUX__Adc0tcomp0ToTimer2trig
enumerator kINPUTMUX__Adc0tcomp1ToTimer2trig
enumerator kINPUTMUX__Adc0tcomp2ToTimer2trig
enumerator kINPUTMUX__Adc0tcomp3ToTimer2trig
enumerator kINPUTMUX__Cmp0outToTimer2trig
enumerator kINPUTMUX__Ctimer0mat1ToTimer2trig
enumerator kINPUTMUX__Ctimer0mat2ToTimer2trig
enumerator kINPUTMUX__Ctimer0mat3ToTimer2trig
enumerator kINPUTMUX__Ctimer1mat1ToTimer2trig
enumerator kINPUTMUX__Ctimer1mat2ToTimer2trig
enumerator kINPUTMUX__Ctimer1mat3ToTimer2trig

enumerator kINPUTMUX_Lpi2c0masterendofpacketToTimer2trig
enumerator kINPUTMUX_Lpi2c0slaveendofpacketToTimer2trig
enumerator kINPUTMUX_Lpi2c1masterendofpacketToTimer2trig
enumerator kINPUTMUX_Lpi2c1slaveendofpacketToTimer2trig
enumerator kINPUTMUX_Lpspi0endofframeToTimer2trig
enumerator kINPUTMUX_Lpspi0receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpspi1endofframeToTimer2trig
enumerator kINPUTMUX_Lpspi1receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart0receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart0transmitteddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart0receivelineidleToTimer2trig
enumerator kINPUTMUX_Lpuart1receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart1transmitteddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart1receivelineidleToTimer2trig
enumerator kINPUTMUX_Lpuart2receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart2transmitteddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart2receivelineidleToTimer2trig
enumerator kINPUTMUX_Lpuart3receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart3transmitteddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart3receivelineidleToTimer2trig
enumerator kINPUTMUX_Lpuart4receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart4transmitteddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart4receivelineidleToTimer2trig
enumerator kINPUTMUX_Adc1tcomp0ToTimer2trig
enumerator kINPUTMUX_Adc1tcomp1ToTimer2trig
enumerator kINPUTMUX_Adc1tcomp2ToTimer2trig
enumerator kINPUTMUX_Adc1tcomp3ToTimer2trig
enumerator kINPUTMUX_Ctimer3mat1ToTimer2trig
enumerator kINPUTMUX_Ctimer3mat2ToTimer2trig
enumerator kINPUTMUX_Ctimer3mat3ToTimer2trig
enumerator kINPUTMUX_Ctimer4mat1ToTimer2trig
enumerator kINPUTMUX_Ctimer4mat2ToTimer2trig
enumerator kINPUTMUX_Ctimer4mat3ToTimer2trig

enumerator kINPUTMUX_Lpi2c2masterendofpacketToTimer2trig
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToTimer2trig
enumerator kINPUTMUX_Lpi2c3masterendofpacketToTimer2trig
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToTimer2trig
enumerator kINPUTMUX_Lpuart5receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart5transmitteddatawordToTimer2trig
enumerator kINPUTMUX_Lpuart5receivelineidleToTimer2trig
enumerator kINPUTMUX_Trigin0ToTimer2trig
enumerator kINPUTMUX_Trigin1ToTimer2trig
enumerator kINPUTMUX_Trigin2ToTimer2trig
enumerator kINPUTMUX_Trigin3ToTimer2trig
enumerator kINPUTMUX_Trigin4ToTimer2trig
enumerator kINPUTMUX_Trigin5ToTimer2trig
enumerator kINPUTMUX_Trigin6ToTimer2trig
enumerator kINPUTMUX_Trigin7ToTimer2trig
enumerator kINPUTMUX_Trigin8ToTimer2trig
enumerator kINPUTMUX_Trigin9ToTimer2trig
enumerator kINPUTMUX_Trigin10ToTimer2trig
enumerator kINPUTMUX_Trigin11ToTimer2trig
enumerator kINPUTMUX_Usb1startofframeToTimer2trig
enumerator kINPUTMUX_Lpspi2endofframeToTimer2trig
enumerator kINPUTMUX_Lpspi2receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpspi3endofframeToTimer2trig
enumerator kINPUTMUX_Lpspi3receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpspi4endofframeToTimer2trig
enumerator kINPUTMUX_Lpspi4receiveddatawordToTimer2trig
enumerator kINPUTMUX_Lpspi5endofframeToTimer2trig
enumerator kINPUTMUX_Lpspi5receiveddatawordToTimer2trig
enumerator kINPUTMUX_Ctinp0ToTimer3trig
enumerator kINPUTMUX_Ctinp1ToTimer3trig
enumerator kINPUTMUX_Ctinp2ToTimer3trig
enumerator kINPUTMUX_Ctinp3ToTimer3trig
enumerator kINPUTMUX_Ctinp4ToTimer3trig

enumerator kINPUTMUX__Ctinp5ToTimer3trig
enumerator kINPUTMUX__Ctinp6ToTimer3trig
enumerator kINPUTMUX__Ctinp7ToTimer3trig
enumerator kINPUTMUX__Ctinp8ToTimer3trig
enumerator kINPUTMUX__Ctinp9ToTimer3trig
enumerator kINPUTMUX__Ctinp10ToTimer3trig
enumerator kINPUTMUX__Ctinp11ToTimer3trig
enumerator kINPUTMUX__Ctinp12ToTimer3trig
enumerator kINPUTMUX__Ctinp13ToTimer3trig
enumerator kINPUTMUX__Ctinp14ToTimer3trig
enumerator kINPUTMUX__Ctinp15ToTimer3trig
enumerator kINPUTMUX__Ctinp16ToTimer3trig
enumerator kINPUTMUX__Ctinp17ToTimer3trig
enumerator kINPUTMUX__Ctinp18ToTimer3trig
enumerator kINPUTMUX__Ctinp19ToTimer3trig
enumerator kINPUTMUX__Aoi0out0ToTimer3trig
enumerator kINPUTMUX__Aoi0out1ToTimer3trig
enumerator kINPUTMUX__Aoi0out2ToTimer3trig
enumerator kINPUTMUX__Aoi0out3ToTimer3trig
enumerator kINPUTMUX__Adc0tcomp0ToTimer3trig
enumerator kINPUTMUX__Adc0tcomp1ToTimer3trig
enumerator kINPUTMUX__Adc0tcomp2ToTimer3trig
enumerator kINPUTMUX__Adc0tcomp3ToTimer3trig
enumerator kINPUTMUX__Cmp0outToTimer3trig
enumerator kINPUTMUX__Ctimer0mat1ToTimer3trig
enumerator kINPUTMUX__Ctimer0mat2ToTimer3trig
enumerator kINPUTMUX__Ctimer0mat3ToTimer3trig
enumerator kINPUTMUX__Ctimer1mat1ToTimer3trig
enumerator kINPUTMUX__Ctimer1mat2ToTimer3trig
enumerator kINPUTMUX__Ctimer1mat3ToTimer3trig
enumerator kINPUTMUX__Lpi2c0masterendofpacketToTimer3trig
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToTimer3trig
enumerator kINPUTMUX__Lpi2c1masterendofpacketToTimer3trig

enumerator kINPUTMUX_Lpi2c1slaveendofpacketToTimer3trig
enumerator kINPUTMUX_Lpspi0endofframeToTimer3trig
enumerator kINPUTMUX_Lpspi0receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpspi1endofframeToTimer3trig
enumerator kINPUTMUX_Lpspi1receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart0receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart0transmitteddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart0receivelineidleToTimer3trig
enumerator kINPUTMUX_Lpuart1receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart1transmitteddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart1receivelineidleToTimer3trig
enumerator kINPUTMUX_Lpuart2receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart2transmitteddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart2receivelineidleToTimer3trig
enumerator kINPUTMUX_Lpuart3receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart3transmitteddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart3receivelineidleToTimer3trig
enumerator kINPUTMUX_Lpuart4receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart4transmitteddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart4receivelineidleToTimer3trig
enumerator kINPUTMUX_Adc1tcomp0ToTimer3trig
enumerator kINPUTMUX_Adc1tcomp1ToTimer3trig
enumerator kINPUTMUX_Adc1tcomp2ToTimer3trig
enumerator kINPUTMUX_Adc1tcomp3ToTimer3trig
enumerator kINPUTMUX_Ctimer2mat1ToTimer3trig
enumerator kINPUTMUX_Ctimer2mat2ToTimer3trig
enumerator kINPUTMUX_Ctimer2mat3ToTimer3trig
enumerator kINPUTMUX_Ctimer4mat1ToTimer3trig
enumerator kINPUTMUX_Ctimer4mat2ToTimer3trig
enumerator kINPUTMUX_Ctimer4mat3ToTimer3trig
enumerator kINPUTMUX_Lpi2c2masterendofpacketToTimer3trig
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToTimer3trig
enumerator kINPUTMUX_Lpi2c3masterendofpacketToTimer3trig

enumerator kINPUTMUX_Lpi2c3slaveendofpacketToTimer3trig
enumerator kINPUTMUX_Lpuart5receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart5transmitteddatawordToTimer3trig
enumerator kINPUTMUX_Lpuart5receivelineidleToTimer3trig
enumerator kINPUTMUX_Tmprout0ToTimer3trig
enumerator kINPUTMUX_Tmprout1ToTimer3trig
enumerator kINPUTMUX_Trigin0ToTimer3trig
enumerator kINPUTMUX_Trigin1ToTimer3trig
enumerator kINPUTMUX_Trigin2ToTimer3trig
enumerator kINPUTMUX_Trigin3ToTimer3trig
enumerator kINPUTMUX_Trigin4ToTimer3trig
enumerator kINPUTMUX_Trigin5ToTimer3trig
enumerator kINPUTMUX_Trigin6ToTimer3trig
enumerator kINPUTMUX_Trigin7ToTimer3trig
enumerator kINPUTMUX_Trigin8ToTimer3trig
enumerator kINPUTMUX_Trigin9ToTimer3trig
enumerator kINPUTMUX_Trigin10ToTimer3trig
enumerator kINPUTMUX_Trigin11ToTimer3trig
enumerator kINPUTMUX_Usb1startofframeToTimer3trig
enumerator kINPUTMUX_Lpspi2endofframeToTimer3trig
enumerator kINPUTMUX_Lpspi2receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpspi3endofframeToTimer3trig
enumerator kINPUTMUX_Lpspi3receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpspi4endofframeToTimer3trig
enumerator kINPUTMUX_Lpspi4receiveddatawordToTimer3trig
enumerator kINPUTMUX_Lpspi5endofframeToTimer3trig
enumerator kINPUTMUX_Lpspi5receiveddatawordToTimer3trig
enumerator kINPUTMUX_Ctinp0ToTimer4trig
enumerator kINPUTMUX_Ctinp1ToTimer4trig
enumerator kINPUTMUX_Ctinp2ToTimer4trig
enumerator kINPUTMUX_Ctinp3ToTimer4trig
enumerator kINPUTMUX_Ctinp4ToTimer4trig
enumerator kINPUTMUX_Ctinp5ToTimer4trig

enumerator kINPUTMUX__Ctinp6ToTimer4trig
enumerator kINPUTMUX__Ctinp7ToTimer4trig
enumerator kINPUTMUX__Ctinp8ToTimer4trig
enumerator kINPUTMUX__Ctinp9ToTimer4trig
enumerator kINPUTMUX__Ctinp10ToTimer4trig
enumerator kINPUTMUX__Ctinp11ToTimer4trig
enumerator kINPUTMUX__Ctinp12ToTimer4trig
enumerator kINPUTMUX__Ctinp13ToTimer4trig
enumerator kINPUTMUX__Ctinp14ToTimer4trig
enumerator kINPUTMUX__Ctinp15ToTimer4trig
enumerator kINPUTMUX__Ctinp16ToTimer4trig
enumerator kINPUTMUX__Ctinp17ToTimer4trig
enumerator kINPUTMUX__Ctinp18ToTimer4trig
enumerator kINPUTMUX__Ctinp19ToTimer4trig
enumerator kINPUTMUX__Aoi0out0ToTimer4trig
enumerator kINPUTMUX__Aoi0out1ToTimer4trig
enumerator kINPUTMUX__Aoi0out2ToTimer4trig
enumerator kINPUTMUX__Aoi0out3ToTimer4trig
enumerator kINPUTMUX__Adc0tcomp0ToTimer4trig
enumerator kINPUTMUX__Adc0tcomp1ToTimer4trig
enumerator kINPUTMUX__Adc0tcomp2ToTimer4trig
enumerator kINPUTMUX__Adc0tcomp3ToTimer4trig
enumerator kINPUTMUX__Cmp0outToTimer4trig
enumerator kINPUTMUX__Ctimer0mat1ToTimer4trig
enumerator kINPUTMUX__Ctimer0mat2ToTimer4trig
enumerator kINPUTMUX__Ctimer0mat3ToTimer4trig
enumerator kINPUTMUX__Ctimer1mat1ToTimer4trig
enumerator kINPUTMUX__Ctimer1mat2ToTimer4trig
enumerator kINPUTMUX__Ctimer1mat3ToTimer4trig
enumerator kINPUTMUX__Lpi2c0masterendofpacketToTimer4trig
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToTimer4trig
enumerator kINPUTMUX__Lpi2c1masterendofpacketToTimer4trig
enumerator kINPUTMUX__Lpi2c1slaveendofpacketToTimer4trig

enumerator kINPUTMUX_Lpspi0endofframeToTimer4trig
enumerator kINPUTMUX_Lpspi0receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpspi1endofframeToTimer4trig
enumerator kINPUTMUX_Lpspi1receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart0receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart0transmitteddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart0receivelineidleToTimer4trig
enumerator kINPUTMUX_Lpuart1receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart1transmitteddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart1receivelineidleToTimer4trig
enumerator kINPUTMUX_Lpuart2receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart2transmitteddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart2receivelineidleToTimer4trig
enumerator kINPUTMUX_Lpuart3receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart3transmitteddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart3receivelineidleToTimer4trig
enumerator kINPUTMUX_Lpuart4receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart4transmitteddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart4receivelineidleToTimer4trig
enumerator kINPUTMUX_Adc1tcomp0ToTimer4trig
enumerator kINPUTMUX_Adc1tcomp1ToTimer4trig
enumerator kINPUTMUX_Adc1tcomp2ToTimer4trig
enumerator kINPUTMUX_Adc1tcomp3ToTimer4trig
enumerator kINPUTMUX_Ctimer2mat1ToTimer4trig
enumerator kINPUTMUX_Ctimer2mat2ToTimer4trig
enumerator kINPUTMUX_Ctimer2mat3ToTimer4trig
enumerator kINPUTMUX_Ctimer3mat1ToTimer4trig
enumerator kINPUTMUX_Ctimer3mat2ToTimer4trig
enumerator kINPUTMUX_Ctimer3mat3ToTimer4trig
enumerator kINPUTMUX_Lpi2c2masterendofpacketToTimer4trig
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToTimer4trig
enumerator kINPUTMUX_Lpi2c3masterendofpacketToTimer4trig
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToTimer4trig

enumerator kINPUTMUX_Lpuart5receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart5transmitteddatawordToTimer4trig
enumerator kINPUTMUX_Lpuart5receivelineidleToTimer4trig
enumerator kINPUTMUX_Trigin0ToTimer4trig
enumerator kINPUTMUX_Trigin1ToTimer4trig
enumerator kINPUTMUX_Trigin2ToTimer4trig
enumerator kINPUTMUX_Trigin3ToTimer4trig
enumerator kINPUTMUX_Trigin4ToTimer4trig
enumerator kINPUTMUX_Trigin5ToTimer4trig
enumerator kINPUTMUX_Trigin6ToTimer4trig
enumerator kINPUTMUX_Trigin7ToTimer4trig
enumerator kINPUTMUX_Trigin8ToTimer4trig
enumerator kINPUTMUX_Trigin9ToTimer4trig
enumerator kINPUTMUX_Trigin10ToTimer4trig
enumerator kINPUTMUX_Trigin11ToTimer4trig
enumerator kINPUTMUX_Usb1startofframeToTimer4trig
enumerator kINPUTMUX_Lpspi2endofframeToTimer4trig
enumerator kINPUTMUX_Lpspi2receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpspi3endofframeToTimer4trig
enumerator kINPUTMUX_Lpspi3receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpspi4endofframeToTimer4trig
enumerator kINPUTMUX_Lpspi4receiveddatawordToTimer4trig
enumerator kINPUTMUX_Lpspi5endofframeToTimer4trig
enumerator kINPUTMUX_Lpspi5receiveddatawordToTimer4trig
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_Aoi0out0ToFreqmeasref

enumerator kINPUTMUX__Aoi0out1ToFreqmeasref
enumerator kINPUTMUX__ClkoutpinToFreqmeasref
enumerator kINPUTMUX__MtrbistheardbeatToFreqmeasref
enumerator kINPUTMUX__SpcbgaprefreshrateinlpmodeToFreqmeasref
enumerator kINPUTMUX__SptimerclockToFreqmeasref
enumerator kINPUTMUX__Spctestobservesfabit0ToFreqmeasref
enumerator kINPUTMUX__SpcpmcobstestdelayxorToFreqmeasref
enumerator kINPUTMUX__Usbp11480mclockdivby20ToFreqmeasref
enumerator kINPUTMUX__Vbatosc32ktesttriggerToFreqmeasref
enumerator kINPUTMUX__Vbatosc32ktestclockToFreqmeasref
enumerator kINPUTMUX__Pll1clkdivToFreqmeasref
enumerator kINPUTMUX__Gdet0onecyclepulseforerrorsdetectedbydet1ToFreqmeasref
enumerator kINPUTMUX__Gdet0onecycleblindingpulseattrimchangeToFreqmeasref
enumerator kINPUTMUX__TsiclkmainlvToFreqmeasref
enumerator kINPUTMUX__TsiclkslvToFreqmeasref
enumerator kINPUTMUX__TsiintgdonelvToFreqmeasref
enumerator kINPUTMUX__TsiintgresetlvToFreqmeasref
enumerator kINPUTMUX__ClkinToFreqmeastar
enumerator kINPUTMUX__Froosc12mToFreqmeastar
enumerator kINPUTMUX__FrohfdvToFreqmeastar
enumerator kINPUTMUX__Osc32k1ToFreqmeastar
enumerator kINPUTMUX__Clk16k1ToFreqmeastar
enumerator kINPUTMUX__SlowclkToFreqmeastar
enumerator kINPUTMUX__Freqmeclkin0ToFreqmeastar
enumerator kINPUTMUX__Freqmeclkin1ToFreqmeastar
enumerator kINPUTMUX__Aoi0out0ToFreqmeastar
enumerator kINPUTMUX__Aoi0out1ToFreqmeastar
enumerator kINPUTMUX__ClkoutpinToFreqmeastar
enumerator kINPUTMUX__MtrbistheardbeatToFreqmeastar
enumerator kINPUTMUX__SpcbgaprefreshrateinlpmodeToFreqmeastar
enumerator kINPUTMUX__SptimerclockToFreqmeastar
enumerator kINPUTMUX__Spctestobservesfabit0ToFreqmeastar
enumerator kINPUTMUX__SpcpmcobstestdelayxorToFreqmeastar

enumerator kINPUTMUX__Usbpll480mclockdivby20ToFreqmeastar
enumerator kINPUTMUX__Vbatosc32ktesttriggerToFreqmeastar
enumerator kINPUTMUX__Vbatosc32ktestclockToFreqmeastar
enumerator kINPUTMUX__Pll1clkdivToFreqmeastar
enumerator kINPUTMUX__Gdet0onecyclepulseforerrorsdetectedbydet1ToFreqmeastar
enumerator kINPUTMUX__Gdet0onecycleblindingpulseattrimchangeToFreqmeastar
enumerator kINPUTMUX__TsiclkmainlvToFreqmeastar
enumerator kINPUTMUX__TsiclkswlvToFreqmeastar
enumerator kINPUTMUX__TsiintgdonelvToFreqmeastar
enumerator kINPUTMUX__TsiintgresetlvToFreqmeastar
enumerator kINPUTMUX__Aoi0out0ToCmp0trig
enumerator kINPUTMUX__Aoi0out1ToCmp0trig
enumerator kINPUTMUX__Aoi0out2ToCmp0trig
enumerator kINPUTMUX__Aoi0out3ToCmp0trig
enumerator kINPUTMUX__Ctimer0mat0ToCmp0trig
enumerator kINPUTMUX__Ctimer0mat2ToCmp0trig
enumerator kINPUTMUX__Ctimer1mat0ToCmp0trig
enumerator kINPUTMUX__Ctimer1mat2ToCmp0trig
enumerator kINPUTMUX__Ctimer2mat0ToCmp0trig
enumerator kINPUTMUX__Ctimer2mat2ToCmp0trig
enumerator kINPUTMUX__Lptmr0ToCmp0trig
enumerator kINPUTMUX__Gpio0pineventtrig0ToCmp0trig
enumerator kINPUTMUX__Gpio1pineventtrig0ToCmp0trig
enumerator kINPUTMUX__Gpio2pineventtrig0ToCmp0trig
enumerator kINPUTMUX__Gpio3pineventtrig0ToCmp0trig
enumerator kINPUTMUX__Gpio4pineventtrig0ToCmp0trig
enumerator kINPUTMUX__WuuToCmp0trig
enumerator kINPUTMUX__Ctimer3mat0ToCmp0trig
enumerator kINPUTMUX__Ctimer3mat1ToCmp0trig
enumerator kINPUTMUX__Ctimer4mat0ToCmp0trig
enumerator kINPUTMUX__Ctimer4mat1ToCmp0trig
enumerator kINPUTMUX__Gpio0pineventtrig1ToCmp0trig
enumerator kINPUTMUX__Gpio1pineventtrig1ToCmp0trig

enumerator kINPUTMUX_Gpio2pineventtrig1ToCmp0trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToCmp0trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToCmp0trig
enumerator kINPUTMUX_ArmtxevToAdc0trig
enumerator kINPUTMUX_Aoi0out0ToAdc0trig
enumerator kINPUTMUX_Aoi0out1ToAdc0trig
enumerator kINPUTMUX_Aoi0out2ToAdc0trig
enumerator kINPUTMUX_Aoi0out3ToAdc0trig
enumerator kINPUTMUX_Cmp0outToAdc0trig
enumerator kINPUTMUX_Ctimer0mat0ToAdc0trig
enumerator kINPUTMUX_Ctimer0mat1ToAdc0trig
enumerator kINPUTMUX_Ctimer1mat0ToAdc0trig
enumerator kINPUTMUX_Ctimer1mat1ToAdc0trig
enumerator kINPUTMUX_Ctimer2mat0ToAdc0trig
enumerator kINPUTMUX_Ctimer2mat1ToAdc0trig
enumerator kINPUTMUX_Lptmr0ToAdc0trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToAdc0trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToAdc0trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToAdc0trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToAdc0trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToAdc0trig
enumerator kINPUTMUX_WuuToAdc0trig
enumerator kINPUTMUX_Adc1tcomp0ToAdc0trig
enumerator kINPUTMUX_Adc1tcomp1ToAdc0trig
enumerator kINPUTMUX_Adc1tcomp2ToAdc0trig
enumerator kINPUTMUX_Adc1tcomp3ToAdc0trig
enumerator kINPUTMUX_Ctimer3mat0ToAdc0trig
enumerator kINPUTMUX_Ctimer3mat1ToAdc0trig
enumerator kINPUTMUX_Ctimer4mat0ToAdc0trig
enumerator kINPUTMUX_Ctimer4mat1ToAdc0trig
enumerator kINPUTMUX_Flexio0ch0ToAdc0trig
enumerator kINPUTMUX_Flexio0ch1ToAdc0trig
enumerator kINPUTMUX_Flexio0ch2ToAdc0trig

enumerator kINPUTMUX_Flexio0ch3ToAdc0trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToAdc0trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToAdc0trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToAdc0trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToAdc0trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToAdc0trig
enumerator kINPUTMUX_ArmtxevToAdc1trig
enumerator kINPUTMUX_Aoi0out0ToAdc1trig
enumerator kINPUTMUX_Aoi0out1ToAdc1trig
enumerator kINPUTMUX_Aoi0out2ToAdc1trig
enumerator kINPUTMUX_Aoi0out3ToAdc1trig
enumerator kINPUTMUX_Cmp0outToAdc1trig
enumerator kINPUTMUX_Ctimer0mat0ToAdc1trig
enumerator kINPUTMUX_Ctimer0mat1ToAdc1trig
enumerator kINPUTMUX_Ctimer1mat0ToAdc1trig
enumerator kINPUTMUX_Ctimer1mat1ToAdc1trig
enumerator kINPUTMUX_Ctimer2mat0ToAdc1trig
enumerator kINPUTMUX_Ctimer2mat1ToAdc1trig
enumerator kINPUTMUX_Lptmr0ToAdc1trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToAdc1trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToAdc1trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToAdc1trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToAdc1trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToAdc1trig
enumerator kINPUTMUX_WuuToAdc1trig
enumerator kINPUTMUX_Adc0tcomp0ToAdc1trig
enumerator kINPUTMUX_Adc0tcomp1ToAdc1trig
enumerator kINPUTMUX_Adc0tcomp2ToAdc1trig
enumerator kINPUTMUX_Adc0tcomp3ToAdc1trig
enumerator kINPUTMUX_Ctimer3mat0ToAdc1trig
enumerator kINPUTMUX_Ctimer3mat1ToAdc1trig
enumerator kINPUTMUX_Ctimer4mat0ToAdc1trig
enumerator kINPUTMUX_Ctimer4mat1ToAdc1trig

enumerator kINPUTMUX_Flexio0ch0ToAdc1trig
enumerator kINPUTMUX_Flexio0ch1ToAdc1trig
enumerator kINPUTMUX_Flexio0ch2ToAdc1trig
enumerator kINPUTMUX_Flexio0ch3ToAdc1trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToAdc1trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToAdc1trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToAdc1trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToAdc1trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToAdc1trig
enumerator kINPUTMUX_Adc0tcomp0ToAoi0input
enumerator kINPUTMUX_Adc0tcomp1ToAoi0input
enumerator kINPUTMUX_Adc0tcomp2ToAoi0input
enumerator kINPUTMUX_Adc0tcomp3ToAoi0input
enumerator kINPUTMUX_Cmp0outToAoi0input
enumerator kINPUTMUX_Ctimer0mat0ToAoi0input
enumerator kINPUTMUX_Ctimer0mat1ToAoi0input
enumerator kINPUTMUX_Ctimer0mat2ToAoi0input
enumerator kINPUTMUX_Ctimer0mat3ToAoi0input
enumerator kINPUTMUX_Ctimer1mat0ToAoi0input
enumerator kINPUTMUX_Ctimer1mat1ToAoi0input
enumerator kINPUTMUX_Ctimer1mat2ToAoi0input
enumerator kINPUTMUX_Ctimer1mat3ToAoi0input
enumerator kINPUTMUX_Ctimer2mat0ToAoi0input
enumerator kINPUTMUX_Ctimer2mat1ToAoi0input
enumerator kINPUTMUX_Ctimer2mat2ToAoi0input
enumerator kINPUTMUX_Ctimer2mat3ToAoi0input
enumerator kINPUTMUX_Lptmr0ToAoi0input
enumerator kINPUTMUX_Trigin0ToAoi0input
enumerator kINPUTMUX_Trigin1ToAoi0input
enumerator kINPUTMUX_Trigin2ToAoi0input
enumerator kINPUTMUX_Trigin3ToAoi0input
enumerator kINPUTMUX_Trigin4ToAoi0input
enumerator kINPUTMUX_Trigin5ToAoi0input

enumerator kINPUTMUX__Trigin6ToAoi0input
enumerator kINPUTMUX__Trigin7ToAoi0input
enumerator kINPUTMUX__Trigin8ToAoi0input
enumerator kINPUTMUX__Trigin9ToAoi0input
enumerator kINPUTMUX__Trigin10ToAoi0input
enumerator kINPUTMUX__Trigin11ToAoi0input
enumerator kINPUTMUX__Gpio0pineventtrig0ToAoi0input
enumerator kINPUTMUX__Gpio1pineventtrig0ToAoi0input
enumerator kINPUTMUX__Gpio2pineventtrig0ToAoi0input
enumerator kINPUTMUX__Gpio3pineventtrig0ToAoi0input
enumerator kINPUTMUX__Gpio4pineventtrig0ToAoi0input
enumerator kINPUTMUX__Adc1tcomp0ToAoi0input
enumerator kINPUTMUX__Adc1tcomp1ToAoi0input
enumerator kINPUTMUX__Adc1tcomp2ToAoi0input
enumerator kINPUTMUX__Adc1tcomp3ToAoi0input
enumerator kINPUTMUX__Ctimer3mat0ToAoi0input
enumerator kINPUTMUX__Ctimer3mat1ToAoi0input
enumerator kINPUTMUX__Ctimer3mat2ToAoi0input
enumerator kINPUTMUX__Ctimer3mat3ToAoi0input
enumerator kINPUTMUX__Ctimer4mat0ToAoi0input
enumerator kINPUTMUX__Ctimer4mat1ToAoi0input
enumerator kINPUTMUX__Ctimer4mat2ToAoi0input
enumerator kINPUTMUX__Ctimer4mat3ToAoi0input
enumerator kINPUTMUX__Flexio0ch0ToAoi0input
enumerator kINPUTMUX__Flexio0ch1ToAoi0input
enumerator kINPUTMUX__Flexio0ch2ToAoi0input
enumerator kINPUTMUX__Flexio0ch3ToAoi0input
enumerator kINPUTMUX__Gpio0pineventtrig1ToAoi0input
enumerator kINPUTMUX__Gpio1pineventtrig1ToAoi0input
enumerator kINPUTMUX__Gpio2pineventtrig1ToAoi0input
enumerator kINPUTMUX__Gpio3pineventtrig1ToAoi0input
enumerator kINPUTMUX__Gpio4pineventtrig1ToAoi0input
enumerator kINPUTMUX__Aoi0out0ToLpi2c0trig

enumerator kINPUTMUX__Aoi0out1ToLpi2c0trig
enumerator kINPUTMUX__Aoi0out2ToLpi2c0trig
enumerator kINPUTMUX__Aoi0out3ToLpi2c0trig
enumerator kINPUTMUX__Cmp0outToLpi2c0trig
enumerator kINPUTMUX__Ctimer0mat0ToLpi2c0trig
enumerator kINPUTMUX__Ctimer0mat1ToLpi2c0trig
enumerator kINPUTMUX__Ctimer1mat0ToLpi2c0trig
enumerator kINPUTMUX__Ctimer1mat1ToLpi2c0trig
enumerator kINPUTMUX__Ctimer2mat0ToLpi2c0trig
enumerator kINPUTMUX__Ctimer2mat1ToLpi2c0trig
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__Gpio4pineventtrig0ToLpi2c0trig
enumerator kINPUTMUX__WuuToLpi2c0trig
enumerator kINPUTMUX__Ctimer3mat2ToLpi2c0trig
enumerator kINPUTMUX__Ctimer3mat3ToLpi2c0trig
enumerator kINPUTMUX__Ctimer4mat2ToLpi2c0trig
enumerator kINPUTMUX__Ctimer4mat3ToLpi2c0trig
enumerator kINPUTMUX__Flexio0ch0ToLpi2c0trig
enumerator kINPUTMUX__Flexio0ch1ToLpi2c0trig
enumerator kINPUTMUX__Flexio0ch2ToLpi2c0trig
enumerator kINPUTMUX__Flexio0ch3ToLpi2c0trig

enumerator kINPUTMUX_Gpio0pineventtrig1ToLpi2c0trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpi2c0trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpi2c0trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpi2c0trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpi2c0trig
enumerator kINPUTMUX_Aoi0out0ToLpi2c1trig
enumerator kINPUTMUX_Aoi0out1ToLpi2c1trig
enumerator kINPUTMUX_Aoi0out2ToLpi2c1trig
enumerator kINPUTMUX_Aoi0out3ToLpi2c1trig
enumerator kINPUTMUX_Cmp0outToLpi2c1trig
enumerator kINPUTMUX_Ctimer0mat0ToLpi2c1trig
enumerator kINPUTMUX_Ctimer0mat1ToLpi2c1trig
enumerator kINPUTMUX_Ctimer1mat0ToLpi2c1trig
enumerator kINPUTMUX_Ctimer1mat1ToLpi2c1trig
enumerator kINPUTMUX_Ctimer2mat0ToLpi2c1trig
enumerator kINPUTMUX_Ctimer2mat1ToLpi2c1trig
enumerator kINPUTMUX_Lptmr0ToLpi2c1trig
enumerator kINPUTMUX_Trigin0ToLpi2c1trig
enumerator kINPUTMUX_Trigin1ToLpi2c1trig
enumerator kINPUTMUX_Trigin2ToLpi2c1trig
enumerator kINPUTMUX_Trigin3ToLpi2c1trig
enumerator kINPUTMUX_Trigin4ToLpi2c1trig
enumerator kINPUTMUX_Trigin5ToLpi2c1trig
enumerator kINPUTMUX_Trigin6ToLpi2c1trig
enumerator kINPUTMUX_Trigin7ToLpi2c1trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToLpi2c1trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToLpi2c1trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToLpi2c1trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToLpi2c1trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToLpi2c1trig
enumerator kINPUTMUX_WuuToLpi2c1trig
enumerator kINPUTMUX_Ctimer3mat2ToLpi2c1trig
enumerator kINPUTMUX_Ctimer3mat3ToLpi2c1trig

enumerator kINPUTMUX__Ctimer4mat2ToLpi2c1trig
enumerator kINPUTMUX__Ctimer4mat3ToLpi2c1trig
enumerator kINPUTMUX__Flexio0ch0ToLpi2c1trig
enumerator kINPUTMUX__Flexio0ch1ToLpi2c1trig
enumerator kINPUTMUX__Flexio0ch2ToLpi2c1trig
enumerator kINPUTMUX__Flexio0ch3ToLpi2c1trig
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpi2c1trig
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpi2c1trig
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpi2c1trig
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpi2c1trig
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpi2c1trig
enumerator kINPUTMUX__Aoi0out0ToLpi2c2trig
enumerator kINPUTMUX__Aoi0out1ToLpi2c2trig
enumerator kINPUTMUX__Aoi0out2ToLpi2c2trig
enumerator kINPUTMUX__Aoi0out3ToLpi2c2trig
enumerator kINPUTMUX__Cmp0outToLpi2c2trig
enumerator kINPUTMUX__Ctimer0mat0ToLpi2c2trig
enumerator kINPUTMUX__Ctimer0mat1ToLpi2c2trig
enumerator kINPUTMUX__Ctimer1mat0ToLpi2c2trig
enumerator kINPUTMUX__Ctimer1mat1ToLpi2c2trig
enumerator kINPUTMUX__Ctimer2mat0ToLpi2c2trig
enumerator kINPUTMUX__Ctimer2mat1ToLpi2c2trig
enumerator kINPUTMUX__Lptmr0ToLpi2c2trig
enumerator kINPUTMUX__Trigin0ToLpi2c2trig
enumerator kINPUTMUX__Trigin1ToLpi2c2trig
enumerator kINPUTMUX__Trigin2ToLpi2c2trig
enumerator kINPUTMUX__Trigin3ToLpi2c2trig
enumerator kINPUTMUX__Trigin4ToLpi2c2trig
enumerator kINPUTMUX__Trigin5ToLpi2c2trig
enumerator kINPUTMUX__Trigin6ToLpi2c2trig
enumerator kINPUTMUX__Trigin7ToLpi2c2trig
enumerator kINPUTMUX__Gpio0pineventtrig0ToLpi2c2trig
enumerator kINPUTMUX__Gpio1pineventtrig0ToLpi2c2trig

enumerator kINPUTMUX_Gpio2pineventtrig0ToLpi2c2trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToLpi2c2trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToLpi2c2trig
enumerator kINPUTMUX_WuuToLpi2c2trig
enumerator kINPUTMUX_Ctimer3mat2ToLpi2c2trig
enumerator kINPUTMUX_Ctimer3mat3ToLpi2c2trig
enumerator kINPUTMUX_Ctimer4mat2ToLpi2c2trig
enumerator kINPUTMUX_Ctimer4mat3ToLpi2c2trig
enumerator kINPUTMUX_Flexio0ch0ToLpi2c2trig
enumerator kINPUTMUX_Flexio0ch1ToLpi2c2trig
enumerator kINPUTMUX_Flexio0ch2ToLpi2c2trig
enumerator kINPUTMUX_Flexio0ch3ToLpi2c2trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpi2c2trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpi2c2trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpi2c2trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpi2c2trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpi2c2trig
enumerator kINPUTMUX_Aoi0out0ToLpi2c3trig
enumerator kINPUTMUX_Aoi0out1ToLpi2c3trig
enumerator kINPUTMUX_Aoi0out2ToLpi2c3trig
enumerator kINPUTMUX_Aoi0out3ToLpi2c3trig
enumerator kINPUTMUX_Cmp0outToLpi2c3trig
enumerator kINPUTMUX_Ctimer0mat0ToLpi2c3trig
enumerator kINPUTMUX_Ctimer0mat1ToLpi2c3trig
enumerator kINPUTMUX_Ctimer1mat0ToLpi2c3trig
enumerator kINPUTMUX_Ctimer1mat1ToLpi2c3trig
enumerator kINPUTMUX_Ctimer2mat0ToLpi2c3trig
enumerator kINPUTMUX_Ctimer2mat1ToLpi2c3trig
enumerator kINPUTMUX_Lptmr0ToLpi2c3trig
enumerator kINPUTMUX_Trigin0ToLpi2c3trig
enumerator kINPUTMUX_Trigin1ToLpi2c3trig
enumerator kINPUTMUX_Trigin2ToLpi2c3trig
enumerator kINPUTMUX_Trigin3ToLpi2c3trig

enumerator kINPUTMUX__Trigin4ToLpi2c3trig
enumerator kINPUTMUX__Trigin5ToLpi2c3trig
enumerator kINPUTMUX__Trigin6ToLpi2c3trig
enumerator kINPUTMUX__Trigin7ToLpi2c3trig
enumerator kINPUTMUX__Gpio0pineventtrig0ToLpi2c3trig
enumerator kINPUTMUX__Gpio1pineventtrig0ToLpi2c3trig
enumerator kINPUTMUX__Gpio2pineventtrig0ToLpi2c3trig
enumerator kINPUTMUX__Gpio3pineventtrig0ToLpi2c3trig
enumerator kINPUTMUX__Gpio4pineventtrig0ToLpi2c3trig
enumerator kINPUTMUX__WuuToLpi2c3trig
enumerator kINPUTMUX__Ctimer3mat2ToLpi2c3trig
enumerator kINPUTMUX__Ctimer3mat3ToLpi2c3trig
enumerator kINPUTMUX__Ctimer4mat2ToLpi2c3trig
enumerator kINPUTMUX__Ctimer4mat3ToLpi2c3trig
enumerator kINPUTMUX__Flexio0ch0ToLpi2c3trig
enumerator kINPUTMUX__Flexio0ch1ToLpi2c3trig
enumerator kINPUTMUX__Flexio0ch2ToLpi2c3trig
enumerator kINPUTMUX__Flexio0ch3ToLpi2c3trig
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpi2c3trig
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpi2c3trig
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpi2c3trig
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpi2c3trig
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpi2c3trig
enumerator kINPUTMUX__Aoi0out0ToLpi2c4trig
enumerator kINPUTMUX__Aoi0out1ToLpi2c4trig
enumerator kINPUTMUX__Aoi0out2ToLpi2c4trig
enumerator kINPUTMUX__Aoi0out3ToLpi2c4trig
enumerator kINPUTMUX__Cmp0outToLpi2c4trig
enumerator kINPUTMUX__Ctimer0mat0ToLpi2c4trig
enumerator kINPUTMUX__Ctimer0mat1ToLpi2c4trig
enumerator kINPUTMUX__Ctimer1mat0ToLpi2c4trig
enumerator kINPUTMUX__Ctimer1mat1ToLpi2c4trig
enumerator kINPUTMUX__Ctimer2mat0ToLpi2c4trig

enumerator kINPUTMUX__Ctimer2mat1ToLpi2c4trig
enumerator kINPUTMUX__Lptmr0ToLpi2c4trig
enumerator kINPUTMUX__Trigin0ToLpi2c4trig
enumerator kINPUTMUX__Trigin1ToLpi2c4trig
enumerator kINPUTMUX__Trigin2ToLpi2c4trig
enumerator kINPUTMUX__Trigin3ToLpi2c4trig
enumerator kINPUTMUX__Trigin4ToLpi2c4trig
enumerator kINPUTMUX__Trigin5ToLpi2c4trig
enumerator kINPUTMUX__Trigin6ToLpi2c4trig
enumerator kINPUTMUX__Trigin7ToLpi2c4trig
enumerator kINPUTMUX__Gpio0pineventtrig0ToLpi2c4trig
enumerator kINPUTMUX__Gpio1pineventtrig0ToLpi2c4trig
enumerator kINPUTMUX__Gpio2pineventtrig0ToLpi2c4trig
enumerator kINPUTMUX__Gpio3pineventtrig0ToLpi2c4trig
enumerator kINPUTMUX__Gpio4pineventtrig0ToLpi2c4trig
enumerator kINPUTMUX__WuuToLpi2c4trig
enumerator kINPUTMUX__Ctimer3mat2ToLpi2c4trig
enumerator kINPUTMUX__Ctimer3mat3ToLpi2c4trig
enumerator kINPUTMUX__Ctimer4mat2ToLpi2c4trig
enumerator kINPUTMUX__Ctimer4mat3ToLpi2c4trig
enumerator kINPUTMUX__Flexio0ch0ToLpi2c4trig
enumerator kINPUTMUX__Flexio0ch1ToLpi2c4trig
enumerator kINPUTMUX__Flexio0ch2ToLpi2c4trig
enumerator kINPUTMUX__Flexio0ch3ToLpi2c4trig
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpi2c4trig
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpi2c4trig
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpi2c4trig
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpi2c4trig
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpi2c4trig
enumerator kINPUTMUX__Aoi0out0ToLpspi0trig
enumerator kINPUTMUX__Aoi0out1ToLpspi0trig
enumerator kINPUTMUX__Aoi0out2ToLpspi0trig
enumerator kINPUTMUX__Aoi0out3ToLpspi0trig

enumerator kINPUTMUX_Cmp0outToLpspi0trig
enumerator kINPUTMUX_Ctimer0mat1ToLpspi0trig
enumerator kINPUTMUX_Ctimer0mat2ToLpspi0trig
enumerator kINPUTMUX_Ctimer1mat1ToLpspi0trig
enumerator kINPUTMUX_Ctimer1mat2ToLpspi0trig
enumerator kINPUTMUX_Ctimer2mat1ToLpspi0trig
enumerator kINPUTMUX_Ctimer2mat2ToLpspi0trig
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_Gpio4pineventtrig0ToLpspi0trig
enumerator kINPUTMUX_WuuToLpspi0trig
enumerator kINPUTMUX_Ctimer3mat2ToLpspi0trig
enumerator kINPUTMUX_Ctimer3mat3ToLpspi0trig
enumerator kINPUTMUX_Ctimer4mat2ToLpspi0trig
enumerator kINPUTMUX_Ctimer4mat3ToLpspi0trig
enumerator kINPUTMUX_Flexio0ch0ToLpspi0trig
enumerator kINPUTMUX_Flexio0ch1ToLpspi0trig
enumerator kINPUTMUX_Flexio0ch2ToLpspi0trig
enumerator kINPUTMUX_Flexio0ch3ToLpspi0trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpspi0trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpspi0trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpspi0trig

enumerator kINPUTMUX_Gpio3pineventtrig1ToLpspi0trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpspi0trig
enumerator kINPUTMUX_Aoi0out0ToLpspi1trig
enumerator kINPUTMUX_Aoi0out1ToLpspi1trig
enumerator kINPUTMUX_Aoi0out2ToLpspi1trig
enumerator kINPUTMUX_Aoi0out3ToLpspi1trig
enumerator kINPUTMUX_Cmp0outToLpspi1trig
enumerator kINPUTMUX_Ctimer0mat1ToLpspi1trig
enumerator kINPUTMUX_Ctimer0mat2ToLpspi1trig
enumerator kINPUTMUX_Ctimer1mat1ToLpspi1trig
enumerator kINPUTMUX_Ctimer1mat2ToLpspi1trig
enumerator kINPUTMUX_Ctimer2mat1ToLpspi1trig
enumerator kINPUTMUX_Ctimer2mat2ToLpspi1trig
enumerator kINPUTMUX_Lptmr0ToLpspi1trig
enumerator kINPUTMUX_Trigin0ToLpspi1trig
enumerator kINPUTMUX_Trigin1ToLpspi1trig
enumerator kINPUTMUX_Trigin2ToLpspi1trig
enumerator kINPUTMUX_Trigin3ToLpspi1trig
enumerator kINPUTMUX_Trigin4ToLpspi1trig
enumerator kINPUTMUX_Trigin5ToLpspi1trig
enumerator kINPUTMUX_Trigin6ToLpspi1trig
enumerator kINPUTMUX_Trigin7ToLpspi1trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToLpspi1trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToLpspi1trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToLpspi1trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToLpspi1trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToLpspi1trig
enumerator kINPUTMUX_WuuToLpspi1trig
enumerator kINPUTMUX_Ctimer3mat2ToLpspi1trig
enumerator kINPUTMUX_Ctimer3mat3ToLpspi1trig
enumerator kINPUTMUX_Ctimer4mat2ToLpspi1trig
enumerator kINPUTMUX_Ctimer4mat3ToLpspi1trig
enumerator kINPUTMUX_Flexio0ch0ToLpspi1trig

enumerator kINPUTMUX_Flexio0ch1ToLpspi1trig
enumerator kINPUTMUX_Flexio0ch2ToLpspi1trig
enumerator kINPUTMUX_Flexio0ch3ToLpspi1trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpspi1trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpspi1trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpspi1trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpspi1trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpspi1trig
enumerator kINPUTMUX_Aoi0out0ToLpspi2trig
enumerator kINPUTMUX_Aoi0out1ToLpspi2trig
enumerator kINPUTMUX_Aoi0out2ToLpspi2trig
enumerator kINPUTMUX_Aoi0out3ToLpspi2trig
enumerator kINPUTMUX_Cmp0outToLpspi2trig
enumerator kINPUTMUX_Ctimer0mat1ToLpspi2trig
enumerator kINPUTMUX_Ctimer0mat2ToLpspi2trig
enumerator kINPUTMUX_Ctimer1mat1ToLpspi2trig
enumerator kINPUTMUX_Ctimer1mat2ToLpspi2trig
enumerator kINPUTMUX_Ctimer2mat1ToLpspi2trig
enumerator kINPUTMUX_Ctimer2mat2ToLpspi2trig
enumerator kINPUTMUX_Lptmr0ToLpspi2trig
enumerator kINPUTMUX_Trigin0ToLpspi2trig
enumerator kINPUTMUX_Trigin1ToLpspi2trig
enumerator kINPUTMUX_Trigin2ToLpspi2trig
enumerator kINPUTMUX_Trigin3ToLpspi2trig
enumerator kINPUTMUX_Trigin4ToLpspi2trig
enumerator kINPUTMUX_Trigin5ToLpspi2trig
enumerator kINPUTMUX_Trigin6ToLpspi2trig
enumerator kINPUTMUX_Trigin7ToLpspi2trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToLpspi2trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToLpspi2trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToLpspi2trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToLpspi2trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToLpspi2trig

enumerator kINPUTMUX__WuuToLpspi2trig
enumerator kINPUTMUX__Ctimer3mat2ToLpspi2trig
enumerator kINPUTMUX__Ctimer3mat3ToLpspi2trig
enumerator kINPUTMUX__Ctimer4mat2ToLpspi2trig
enumerator kINPUTMUX__Ctimer4mat3ToLpspi2trig
enumerator kINPUTMUX__Flexio0ch0ToLpspi2trig
enumerator kINPUTMUX__Flexio0ch1ToLpspi2trig
enumerator kINPUTMUX__Flexio0ch2ToLpspi2trig
enumerator kINPUTMUX__Flexio0ch3ToLpspi2trig
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpspi2trig
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpspi2trig
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpspi2trig
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpspi2trig
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpspi2trig
enumerator kINPUTMUX__Aoi0out0ToLpspi3trig
enumerator kINPUTMUX__Aoi0out1ToLpspi3trig
enumerator kINPUTMUX__Aoi0out2ToLpspi3trig
enumerator kINPUTMUX__Aoi0out3ToLpspi3trig
enumerator kINPUTMUX__Cmp0outToLpspi3trig
enumerator kINPUTMUX__Ctimer0mat1ToLpspi3trig
enumerator kINPUTMUX__Ctimer0mat2ToLpspi3trig
enumerator kINPUTMUX__Ctimer1mat1ToLpspi3trig
enumerator kINPUTMUX__Ctimer1mat2ToLpspi3trig
enumerator kINPUTMUX__Ctimer2mat1ToLpspi3trig
enumerator kINPUTMUX__Ctimer2mat2ToLpspi3trig
enumerator kINPUTMUX__Lptmr0ToLpspi3trig
enumerator kINPUTMUX__Trigin0ToLpspi3trig
enumerator kINPUTMUX__Trigin1ToLpspi3trig
enumerator kINPUTMUX__Trigin2ToLpspi3trig
enumerator kINPUTMUX__Trigin3ToLpspi3trig
enumerator kINPUTMUX__Trigin4ToLpspi3trig
enumerator kINPUTMUX__Trigin5ToLpspi3trig
enumerator kINPUTMUX__Trigin6ToLpspi3trig

enumerator kINPUTMUX__Trigin7ToLpspi3trig
enumerator kINPUTMUX__Gpio0pineventtrig0ToLpspi3trig
enumerator kINPUTMUX__Gpio1pineventtrig0ToLpspi3trig
enumerator kINPUTMUX__Gpio2pineventtrig0ToLpspi3trig
enumerator kINPUTMUX__Gpio3pineventtrig0ToLpspi3trig
enumerator kINPUTMUX__Gpio4pineventtrig0ToLpspi3trig
enumerator kINPUTMUX__WuuToLpspi3trig
enumerator kINPUTMUX__Ctimer3mat2ToLpspi3trig
enumerator kINPUTMUX__Ctimer3mat3ToLpspi3trig
enumerator kINPUTMUX__Ctimer4mat2ToLpspi3trig
enumerator kINPUTMUX__Ctimer4mat3ToLpspi3trig
enumerator kINPUTMUX__Flexio0ch0ToLpspi3trig
enumerator kINPUTMUX__Flexio0ch1ToLpspi3trig
enumerator kINPUTMUX__Flexio0ch2ToLpspi3trig
enumerator kINPUTMUX__Flexio0ch3ToLpspi3trig
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpspi3trig
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpspi3trig
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpspi3trig
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpspi3trig
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpspi3trig
enumerator kINPUTMUX__Aoi0out0ToLpspi4trig
enumerator kINPUTMUX__Aoi0out1ToLpspi4trig
enumerator kINPUTMUX__Aoi0out2ToLpspi4trig
enumerator kINPUTMUX__Aoi0out3ToLpspi4trig
enumerator kINPUTMUX__Cmp0outToLpspi4trig
enumerator kINPUTMUX__Ctimer0mat1ToLpspi4trig
enumerator kINPUTMUX__Ctimer0mat2ToLpspi4trig
enumerator kINPUTMUX__Ctimer1mat1ToLpspi4trig
enumerator kINPUTMUX__Ctimer1mat2ToLpspi4trig
enumerator kINPUTMUX__Ctimer2mat1ToLpspi4trig
enumerator kINPUTMUX__Ctimer2mat2ToLpspi4trig
enumerator kINPUTMUX__Lptmr0ToLpspi4trig
enumerator kINPUTMUX__Trigin0ToLpspi4trig

enumerator kINPUTMUX__Trigin1ToLpspi4trig
enumerator kINPUTMUX__Trigin2ToLpspi4trig
enumerator kINPUTMUX__Trigin3ToLpspi4trig
enumerator kINPUTMUX__Trigin4ToLpspi4trig
enumerator kINPUTMUX__Trigin5ToLpspi4trig
enumerator kINPUTMUX__Trigin6ToLpspi4trig
enumerator kINPUTMUX__Trigin7ToLpspi4trig
enumerator kINPUTMUX__Gpio0pineventtrig0ToLpspi4trig
enumerator kINPUTMUX__Gpio1pineventtrig0ToLpspi4trig
enumerator kINPUTMUX__Gpio2pineventtrig0ToLpspi4trig
enumerator kINPUTMUX__Gpio3pineventtrig0ToLpspi4trig
enumerator kINPUTMUX__Gpio4pineventtrig0ToLpspi4trig
enumerator kINPUTMUX__WuuToLpspi4trig
enumerator kINPUTMUX__Ctimer3mat2ToLpspi4trig
enumerator kINPUTMUX__Ctimer3mat3ToLpspi4trig
enumerator kINPUTMUX__Ctimer4mat2ToLpspi4trig
enumerator kINPUTMUX__Ctimer4mat3ToLpspi4trig
enumerator kINPUTMUX__Flexio0ch0ToLpspi4trig
enumerator kINPUTMUX__Flexio0ch1ToLpspi4trig
enumerator kINPUTMUX__Flexio0ch2ToLpspi4trig
enumerator kINPUTMUX__Flexio0ch3ToLpspi4trig
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpspi4trig
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpspi4trig
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpspi4trig
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpspi4trig
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpspi4trig
enumerator kINPUTMUX__Aoi0out0ToLpspi5trig
enumerator kINPUTMUX__Aoi0out1ToLpspi5trig
enumerator kINPUTMUX__Aoi0out2ToLpspi5trig
enumerator kINPUTMUX__Aoi0out3ToLpspi5trig
enumerator kINPUTMUX__Cmp0outToLpspi5trig
enumerator kINPUTMUX__Ctimer0mat1ToLpspi5trig
enumerator kINPUTMUX__Ctimer0mat2ToLpspi5trig

enumerator kINPUTMUX_Ctimer1mat1ToLpspi5trig
enumerator kINPUTMUX_Ctimer1mat2ToLpspi5trig
enumerator kINPUTMUX_Ctimer2mat1ToLpspi5trig
enumerator kINPUTMUX_Ctimer2mat2ToLpspi5trig
enumerator kINPUTMUX_Lptmr0ToLpspi5trig
enumerator kINPUTMUX_Trigin0ToLpspi5trig
enumerator kINPUTMUX_Trigin1ToLpspi5trig
enumerator kINPUTMUX_Trigin2ToLpspi5trig
enumerator kINPUTMUX_Trigin3ToLpspi5trig
enumerator kINPUTMUX_Trigin4ToLpspi5trig
enumerator kINPUTMUX_Trigin5ToLpspi5trig
enumerator kINPUTMUX_Trigin6ToLpspi5trig
enumerator kINPUTMUX_Trigin7ToLpspi5trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToLpspi5trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToLpspi5trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToLpspi5trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToLpspi5trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToLpspi5trig
enumerator kINPUTMUX_WuuToLpspi5trig
enumerator kINPUTMUX_Ctimer3mat2ToLpspi5trig
enumerator kINPUTMUX_Ctimer3mat3ToLpspi5trig
enumerator kINPUTMUX_Ctimer4mat2ToLpspi5trig
enumerator kINPUTMUX_Ctimer4mat3ToLpspi5trig
enumerator kINPUTMUX_Flexio0ch0ToLpspi5trig
enumerator kINPUTMUX_Flexio0ch1ToLpspi5trig
enumerator kINPUTMUX_Flexio0ch2ToLpspi5trig
enumerator kINPUTMUX_Flexio0ch3ToLpspi5trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpspi5trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpspi5trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpspi5trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpspi5trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpspi5trig
enumerator kINPUTMUX_Aoi0out0ToLpuart0

enumerator kINPUTMUX__Aoi0out1ToLpuart0
enumerator kINPUTMUX__Aoi0out2ToLpuart0
enumerator kINPUTMUX__Aoi0out3ToLpuart0
enumerator kINPUTMUX__Cmp0outToLpuart0
enumerator kINPUTMUX__Ctimer0mat2ToLpuart0
enumerator kINPUTMUX__Ctimer0mat3ToLpuart0
enumerator kINPUTMUX__Ctimer1mat2ToLpuart0
enumerator kINPUTMUX__Ctimer1mat3ToLpuart0
enumerator kINPUTMUX__Ctimer2mat2ToLpuart0
enumerator kINPUTMUX__Ctimer2mat3ToLpuart0
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__Gpio4pineventtrig0ToLpuart0
enumerator kINPUTMUX__WuuToLpuart0
enumerator kINPUTMUX__Ctimer3mat2ToLpuart0
enumerator kINPUTMUX__Ctimer3mat3ToLpuart0
enumerator kINPUTMUX__Ctimer4mat2ToLpuart0
enumerator kINPUTMUX__Ctimer4mat3ToLpuart0

enumerator kINPUTMUX_Flexio0ch0ToLpuart0
enumerator kINPUTMUX_Flexio0ch1ToLpuart0
enumerator kINPUTMUX_Flexio0ch2ToLpuart0
enumerator kINPUTMUX_Flexio0ch3ToLpuart0
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpuart0
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpuart0
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpuart0
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpuart0
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpuart0
enumerator kINPUTMUX_Aoi0out0ToLpuart1
enumerator kINPUTMUX_Aoi0out1ToLpuart1
enumerator kINPUTMUX_Aoi0out2ToLpuart1
enumerator kINPUTMUX_Aoi0out3ToLpuart1
enumerator kINPUTMUX_Cmp0outToLpuart1
enumerator kINPUTMUX_Ctimer0mat2ToLpuart1
enumerator kINPUTMUX_Ctimer0mat3ToLpuart1
enumerator kINPUTMUX_Ctimer1mat2ToLpuart1
enumerator kINPUTMUX_Ctimer1mat3ToLpuart1
enumerator kINPUTMUX_Ctimer2mat2ToLpuart1
enumerator kINPUTMUX_Ctimer2mat3ToLpuart1
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_Gpio4pineventtrig0ToLpuart1
enumerator kINPUTMUX_WuuToLpuart1
enumerator kINPUTMUX_Ctimer3mat2ToLpuart1
enumerator kINPUTMUX_Ctimer3mat3ToLpuart1
enumerator kINPUTMUX_Ctimer4mat2ToLpuart1
enumerator kINPUTMUX_Ctimer4mat3ToLpuart1
enumerator kINPUTMUX_Flexio0ch0ToLpuart1
enumerator kINPUTMUX_Flexio0ch1ToLpuart1
enumerator kINPUTMUX_Flexio0ch2ToLpuart1
enumerator kINPUTMUX_Flexio0ch3ToLpuart1
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpuart1
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpuart1
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpuart1
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpuart1
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpuart1
enumerator kINPUTMUX_Aoi0out0ToLpuart2
enumerator kINPUTMUX_Aoi0out1ToLpuart2
enumerator kINPUTMUX_Aoi0out2ToLpuart2
enumerator kINPUTMUX_Aoi0out3ToLpuart2
enumerator kINPUTMUX_Cmp0outToLpuart2
enumerator kINPUTMUX_Ctimer0mat2ToLpuart2
enumerator kINPUTMUX_Ctimer0mat3ToLpuart2
enumerator kINPUTMUX_Ctimer1mat2ToLpuart2
enumerator kINPUTMUX_Ctimer1mat3ToLpuart2
enumerator kINPUTMUX_Ctimer2mat2ToLpuart2
enumerator kINPUTMUX_Ctimer2mat3ToLpuart2
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__Gpio4pineventtrig0ToLpuart2
enumerator kINPUTMUX__WuuToLpuart2
enumerator kINPUTMUX__Ctimer3mat2ToLpuart2
enumerator kINPUTMUX__Ctimer3mat3ToLpuart2
enumerator kINPUTMUX__Ctimer4mat2ToLpuart2
enumerator kINPUTMUX__Ctimer4mat3ToLpuart2
enumerator kINPUTMUX__Flexio0ch0ToLpuart2
enumerator kINPUTMUX__Flexio0ch1ToLpuart2
enumerator kINPUTMUX__Flexio0ch2ToLpuart2
enumerator kINPUTMUX__Flexio0ch3ToLpuart2
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpuart2
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpuart2
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpuart2
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpuart2
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpuart2
enumerator kINPUTMUX__Aoi0out0ToLpuart3
enumerator kINPUTMUX__Aoi0out1ToLpuart3
enumerator kINPUTMUX__Aoi0out2ToLpuart3
enumerator kINPUTMUX__Aoi0out3ToLpuart3

enumerator kINPUTMUX_Cmp0outToLpuart3
enumerator kINPUTMUX_Ctimer0mat2ToLpuart3
enumerator kINPUTMUX_Ctimer0mat3ToLpuart3
enumerator kINPUTMUX_Ctimer1mat2ToLpuart3
enumerator kINPUTMUX_Ctimer1mat3ToLpuart3
enumerator kINPUTMUX_Ctimer2mat2ToLpuart3
enumerator kINPUTMUX_Ctimer2mat3ToLpuart3
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_Gpio4pineventtrig0ToLpuart3
enumerator kINPUTMUX_WuuToLpuart3
enumerator kINPUTMUX_Ctimer3mat2ToLpuart3
enumerator kINPUTMUX_Ctimer3mat3ToLpuart3
enumerator kINPUTMUX_Ctimer4mat2ToLpuart3
enumerator kINPUTMUX_Ctimer4mat3ToLpuart3
enumerator kINPUTMUX_Flexio0ch0ToLpuart3
enumerator kINPUTMUX_Flexio0ch1ToLpuart3
enumerator kINPUTMUX_Flexio0ch2ToLpuart3

enumerator kINPUTMUX_Flexio0ch3ToLpuart3
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpuart3
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpuart3
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpuart3
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpuart3
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpuart3
enumerator kINPUTMUX_Aoi0out0ToLpuart4
enumerator kINPUTMUX_Aoi0out1ToLpuart4
enumerator kINPUTMUX_Aoi0out2ToLpuart4
enumerator kINPUTMUX_Aoi0out3ToLpuart4
enumerator kINPUTMUX_Cmp0outToLpuart4
enumerator kINPUTMUX_Ctimer0mat2ToLpuart4
enumerator kINPUTMUX_Ctimer0mat3ToLpuart4
enumerator kINPUTMUX_Ctimer1mat2ToLpuart4
enumerator kINPUTMUX_Ctimer1mat3ToLpuart4
enumerator kINPUTMUX_Ctimer2mat2ToLpuart4
enumerator kINPUTMUX_Ctimer2mat3ToLpuart4
enumerator kINPUTMUX_Lptmr0ToLpuart4
enumerator kINPUTMUX_Trigin0ToLpuart4
enumerator kINPUTMUX_Trigin1ToLpuart4
enumerator kINPUTMUX_Trigin2ToLpuart4
enumerator kINPUTMUX_Trigin3ToLpuart4
enumerator kINPUTMUX_Trigin4ToLpuart4
enumerator kINPUTMUX_Trigin5ToLpuart4
enumerator kINPUTMUX_Trigin6ToLpuart4
enumerator kINPUTMUX_Trigin7ToLpuart4
enumerator kINPUTMUX_Trigin8ToLpuart4
enumerator kINPUTMUX_Trigin9ToLpuart4
enumerator kINPUTMUX_Trigin10ToLpuart4
enumerator kINPUTMUX_Trigin11ToLpuart4
enumerator kINPUTMUX_Gpio0pineventtrig0ToLpuart4
enumerator kINPUTMUX_Gpio1pineventtrig0ToLpuart4
enumerator kINPUTMUX_Gpio2pineventtrig0ToLpuart4

enumerator kINPUTMUX_Gpio3pineventtrig0ToLpuart4
enumerator kINPUTMUX_Gpio4pineventtrig0ToLpuart4
enumerator kINPUTMUX_WuuToLpuart4
enumerator kINPUTMUX_Ctimer3mat2ToLpuart4
enumerator kINPUTMUX_Ctimer3mat3ToLpuart4
enumerator kINPUTMUX_Ctimer4mat2ToLpuart4
enumerator kINPUTMUX_Ctimer4mat3ToLpuart4
enumerator kINPUTMUX_Flexio0ch0ToLpuart4
enumerator kINPUTMUX_Flexio0ch1ToLpuart4
enumerator kINPUTMUX_Flexio0ch2ToLpuart4
enumerator kINPUTMUX_Flexio0ch3ToLpuart4
enumerator kINPUTMUX_Gpio0pineventtrig1ToLpuart4
enumerator kINPUTMUX_Gpio1pineventtrig1ToLpuart4
enumerator kINPUTMUX_Gpio2pineventtrig1ToLpuart4
enumerator kINPUTMUX_Gpio3pineventtrig1ToLpuart4
enumerator kINPUTMUX_Gpio4pineventtrig1ToLpuart4
enumerator kINPUTMUX_Aoi0out0ToLpuart5
enumerator kINPUTMUX_Aoi0out1ToLpuart5
enumerator kINPUTMUX_Aoi0out2ToLpuart5
enumerator kINPUTMUX_Aoi0out3ToLpuart5
enumerator kINPUTMUX_Cmp0outToLpuart5
enumerator kINPUTMUX_Ctimer0mat2ToLpuart5
enumerator kINPUTMUX_Ctimer0mat3ToLpuart5
enumerator kINPUTMUX_Ctimer1mat2ToLpuart5
enumerator kINPUTMUX_Ctimer1mat3ToLpuart5
enumerator kINPUTMUX_Ctimer2mat2ToLpuart5
enumerator kINPUTMUX_Ctimer2mat3ToLpuart5
enumerator kINPUTMUX_Lptmr0ToLpuart5
enumerator kINPUTMUX_Trigin0ToLpuart5
enumerator kINPUTMUX_Trigin1ToLpuart5
enumerator kINPUTMUX_Trigin2ToLpuart5
enumerator kINPUTMUX_Trigin3ToLpuart5
enumerator kINPUTMUX_Trigin4ToLpuart5

enumerator kINPUTMUX__Trigin5ToLpuart5
enumerator kINPUTMUX__Trigin6ToLpuart5
enumerator kINPUTMUX__Trigin7ToLpuart5
enumerator kINPUTMUX__Trigin8ToLpuart5
enumerator kINPUTMUX__Trigin9ToLpuart5
enumerator kINPUTMUX__Trigin10ToLpuart5
enumerator kINPUTMUX__Trigin11ToLpuart5
enumerator kINPUTMUX__Gpio0pineventtrig0ToLpuart5
enumerator kINPUTMUX__Gpio1pineventtrig0ToLpuart5
enumerator kINPUTMUX__Gpio2pineventtrig0ToLpuart5
enumerator kINPUTMUX__Gpio3pineventtrig0ToLpuart5
enumerator kINPUTMUX__Gpio4pineventtrig0ToLpuart5
enumerator kINPUTMUX__WuuToLpuart5
enumerator kINPUTMUX__Ctimer3mat2ToLpuart5
enumerator kINPUTMUX__Ctimer3mat3ToLpuart5
enumerator kINPUTMUX__Ctimer4mat2ToLpuart5
enumerator kINPUTMUX__Ctimer4mat3ToLpuart5
enumerator kINPUTMUX__Flexio0ch0ToLpuart5
enumerator kINPUTMUX__Flexio0ch1ToLpuart5
enumerator kINPUTMUX__Flexio0ch2ToLpuart5
enumerator kINPUTMUX__Flexio0ch3ToLpuart5
enumerator kINPUTMUX__Gpio0pineventtrig1ToLpuart5
enumerator kINPUTMUX__Gpio1pineventtrig1ToLpuart5
enumerator kINPUTMUX__Gpio2pineventtrig1ToLpuart5
enumerator kINPUTMUX__Gpio3pineventtrig1ToLpuart5
enumerator kINPUTMUX__Gpio4pineventtrig1ToLpuart5
enumerator kINPUTMUX__Aoi0out0ToFlexiotrig
enumerator kINPUTMUX__Aoi0out1ToFlexiotrig
enumerator kINPUTMUX__Aoi0out2ToFlexiotrig
enumerator kINPUTMUX__Aoi0out3ToFlexiotrig
enumerator kINPUTMUX__Adc0tcomp0ToFlexiotrig
enumerator kINPUTMUX__Adc0tcomp1ToFlexiotrig
enumerator kINPUTMUX__Adc0tcomp2ToFlexiotrig

enumerator kINPUTMUX__Adc0tcomp3ToFlexiotrig
enumerator kINPUTMUX__Cmp0outToFlexiotrig
enumerator kINPUTMUX__Ctimer0mat1ToFlexiotrig
enumerator kINPUTMUX__Ctimer0mat2ToFlexiotrig
enumerator kINPUTMUX__Ctimer1mat1ToFlexiotrig
enumerator kINPUTMUX__Ctimer1mat2ToFlexiotrig
enumerator kINPUTMUX__Ctimer2mat1ToFlexiotrig
enumerator kINPUTMUX__Ctimer2mat2ToFlexiotrig
enumerator kINPUTMUX__Lptmr0ToFlexiotrig
enumerator kINPUTMUX__Trigin0ToFlexiotrig
enumerator kINPUTMUX__Trigin1ToFlexiotrig
enumerator kINPUTMUX__Trigin2ToFlexiotrig
enumerator kINPUTMUX__Trigin3ToFlexiotrig
enumerator kINPUTMUX__Trigin4ToFlexiotrig
enumerator kINPUTMUX__Trigin5ToFlexiotrig
enumerator kINPUTMUX__Trigin6ToFlexiotrig
enumerator kINPUTMUX__Trigin7ToFlexiotrig
enumerator kINPUTMUX__Gpio0pineventtrig0ToFlexiotrig
enumerator kINPUTMUX__Gpio1pineventtrig0ToFlexiotrig
enumerator kINPUTMUX__Gpio2pineventtrig0ToFlexiotrig
enumerator kINPUTMUX__Gpio3pineventtrig0ToFlexiotrig
enumerator kINPUTMUX__Gpio4pineventtrig0ToFlexiotrig
enumerator kINPUTMUX__WuuToFlexiotrig
enumerator kINPUTMUX__Lpi2c0masterendofpacketToFlexiotrig
enumerator kINPUTMUX__Lpi2c0slaveendofpacketToFlexiotrig
enumerator kINPUTMUX__Lpi2c1masterendofpacketToFlexiotrig
enumerator kINPUTMUX__Lpi2c1slaveendofpacketToFlexiotrig
enumerator kINPUTMUX__Lpspi0endofframeToFlexiotrig
enumerator kINPUTMUX__Lpspi0receiveddatawordToFlexiotrig
enumerator kINPUTMUX__Lpspi1endofframeToFlexiotrig
enumerator kINPUTMUX__Lpspi1receiveddatawordToFlexiotrig
enumerator kINPUTMUX__Lpuart0receiveddatawordToFlexiotrig
enumerator kINPUTMUX__Lpuart0transmitteddatawordToFlexiotrig

enumerator kINPUTMUX_Lpuart0receivelineidleToFlexiotrig
enumerator kINPUTMUX_Lpuart1receiveddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart1transmitteddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart1receivelineidleToFlexiotrig
enumerator kINPUTMUX_Lpuart2receiveddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart2transmitteddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart2receivelineidleToFlexiotrig
enumerator kINPUTMUX_Lpuart3receiveddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart3transmitteddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart3receivelineidleToFlexiotrig
enumerator kINPUTMUX_Lpuart4receiveddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart4transmitteddatawordToFlexiotrig
enumerator kINPUTMUX_Lpuart4receivelineidleToFlexiotrig
enumerator kINPUTMUX_Adc1tcomp0ToFlexiotrig
enumerator kINPUTMUX_Adc1tcomp1ToFlexiotrig
enumerator kINPUTMUX_Adc1tcomp2ToFlexiotrig
enumerator kINPUTMUX_Adc1tcomp3ToFlexiotrig
enumerator kINPUTMUX_Ctimer3mat2ToFlexiotrig
enumerator kINPUTMUX_Ctimer3mat3ToFlexiotrig
enumerator kINPUTMUX_Ctimer4mat2ToFlexiotrig
enumerator kINPUTMUX_Ctimer4mat3ToFlexiotrig
enumerator kINPUTMUX_Lpi2c2masterendofpacketToFlexiotrig
enumerator kINPUTMUX_Lpi2c2slaveendofpacketToFlexiotrig
enumerator kINPUTMUX_Lpi2c3masterendofpacketToFlexiotrig
enumerator kINPUTMUX_Lpi2c3slaveendofpacketToFlexiotrig
enumerator kINPUTMUX_Lpspi2endofframeToFlexiotrig
enumerator kINPUTMUX_Lpspi2receiveddatawordToFlexiotrig
enumerator kINPUTMUX_Lpspi3endofframeToFlexiotrig
enumerator kINPUTMUX_Lpspi3receiveddatawordToFlexiotrig
enumerator kINPUTMUX_Gpio0pineventtrig1ToFlexiotrig
enumerator kINPUTMUX_Gpio1pineventtrig1ToFlexiotrig
enumerator kINPUTMUX_Gpio2pineventtrig1ToFlexiotrig
enumerator kINPUTMUX_Gpio3pineventtrig1ToFlexiotrig

enumerator kINPUTMUX_Gpio4pineventtrig1ToFlexiotrig
enumerator kINPUTMUX_ArmtxevToDac0trig
enumerator kINPUTMUX_Aoi0out0ToDac0trig
enumerator kINPUTMUX_Aoi0out1ToDac0trig
enumerator kINPUTMUX_Aoi0out2ToDac0trig
enumerator kINPUTMUX_Aoi0out3ToDac0trig
enumerator kINPUTMUX_Cmp0outToDac0trig
enumerator kINPUTMUX_Ctimer0mat0ToDac0trig
enumerator kINPUTMUX_Ctimer0mat1ToDac0trig
enumerator kINPUTMUX_Ctimer1mat0ToDac0trig
enumerator kINPUTMUX_Ctimer1mat1ToDac0trig
enumerator kINPUTMUX_Ctimer2mat0ToDac0trig
enumerator kINPUTMUX_Ctimer2mat1ToDac0trig
enumerator kINPUTMUX_Lptmr0ToDac0trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToDac0trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToDac0trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToDac0trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToDac0trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToDac0trig
enumerator kINPUTMUX_WuuToDac0trig
enumerator kINPUTMUX_Adc0tcomp0ToDac0trig
enumerator kINPUTMUX_Adc0tcomp1ToDac0trig
enumerator kINPUTMUX_Adc1tcomp0ToDac0trig
enumerator kINPUTMUX_Adc1tcomp1ToDac0trig
enumerator kINPUTMUX_Ctimer3mat0ToDac0trig
enumerator kINPUTMUX_Ctimer3mat1ToDac0trig
enumerator kINPUTMUX_Ctimer4mat0ToDac0trig
enumerator kINPUTMUX_Ctimer4mat1ToDac0trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToDac0trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToDac0trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToDac0trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToDac0trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToDac0trig

enumerator kINPUTMUX__Ctimer0mat2ToTsi0triginput
enumerator kINPUTMUX__Ctimer0mat3ToTsi0triginput
enumerator kINPUTMUX__Ctimer1mat2ToTsi0triginput
enumerator kINPUTMUX__Ctimer1mat3ToTsi0triginput
enumerator kINPUTMUX__Ctimer2mat2ToTsi0triginput
enumerator kINPUTMUX__Ctimer2mat3ToTsi0triginput
enumerator kINPUTMUX__Ctimer3mat2ToTsi0triginput
enumerator kINPUTMUX__Ctimer3mat3ToTsi0triginput
enumerator kINPUTMUX__Ctimer4mat2ToTsi0triginput
enumerator kINPUTMUX__Ctimer4mat3ToTsi0triginput
enumerator kINPUTMUX__Lptmr0ToTsi0triginput
enumerator kINPUTMUX__WuuToTsi0triginput
enumerator kINPUTMUX__Tenbaset1srxeventoutputToEnettrigin
enumerator kINPUTMUX__Tenbaset1stxeventoutputToEnettrigin
enumerator kINPUTMUX__Trigin0ToEnettrigin
enumerator kINPUTMUX__Trigin1ToEnettrigin
enumerator kINPUTMUX__Trigin2ToEnettrigin
enumerator kINPUTMUX__Trigin3ToEnettrigin
enumerator kINPUTMUX__Trigin4ToEnettrigin
enumerator kINPUTMUX__Trigin5ToEnettrigin
enumerator kINPUTMUX__Trigin6ToEnettrigin
enumerator kINPUTMUX__Trigin7ToEnettrigin
enumerator kINPUTMUX__Trigin8ToEnettrigin
enumerator kINPUTMUX__Trigin9ToEnettrigin
enumerator kINPUTMUX__Trigin10ToEnettrigin
enumerator kINPUTMUX__Trigin11ToEnettrigin
enumerator kINPUTMUX__Gpio0pineventtrig0ToEnettrigin
enumerator kINPUTMUX__Gpio1pineventtrig0ToEnettrigin
enumerator kINPUTMUX__Gpio2pineventtrig0ToEnettrigin
enumerator kINPUTMUX__Gpio3pineventtrig0ToEnettrigin
enumerator kINPUTMUX__Gpio4pineventtrig0ToEnettrigin
enumerator kINPUTMUX__Gpio0pineventtrig1ToEnettrigin
enumerator kINPUTMUX__Gpio1pineventtrig1ToEnettrigin

enumerator kINPUTMUX_Gpio2pineventtrig1ToEnettrigin
enumerator kINPUTMUX_Gpio3pineventtrig1ToEnettrigin
enumerator kINPUTMUX_Gpio4pineventtrig1ToEnettrigin
enumerator kINPUTMUX_Aoi0out0ToT1swkup
enumerator kINPUTMUX_Aoi0out1ToT1swkup
enumerator kINPUTMUX_Aoi0out2ToT1swkup
enumerator kINPUTMUX_Aoi0out3ToT1swkup
enumerator kINPUTMUX_Adc0tcomp0ToT1swkup
enumerator kINPUTMUX_Adc0tcomp1ToT1swkup
enumerator kINPUTMUX_Adc0tcomp2ToT1swkup
enumerator kINPUTMUX_Adc0tcomp3ToT1swkup
enumerator kINPUTMUX_Cmp0outToT1swkup
enumerator kINPUTMUX_Ctimer0mat1ToT1swkup
enumerator kINPUTMUX_Ctimer0mat2ToT1swkup
enumerator kINPUTMUX_Ctimer1mat1ToT1swkup
enumerator kINPUTMUX_Ctimer1mat2ToT1swkup
enumerator kINPUTMUX_Ctimer2mat1ToT1swkup
enumerator kINPUTMUX_Ctimer2mat2ToT1swkup
enumerator kINPUTMUX_Lptmr0ToT1swkup
enumerator kINPUTMUX_Trigin0ToT1swkup
enumerator kINPUTMUX_Trigin1ToT1swkup
enumerator kINPUTMUX_Trigin2ToT1swkup
enumerator kINPUTMUX_Trigin3ToT1swkup
enumerator kINPUTMUX_Trigin4ToT1swkup
enumerator kINPUTMUX_Trigin5ToT1swkup
enumerator kINPUTMUX_Trigin6ToT1swkup
enumerator kINPUTMUX_Trigin7ToT1swkup
enumerator kINPUTMUX_Gpio0pineventtrig0ToT1swkup
enumerator kINPUTMUX_Gpio1pineventtrig0ToT1swkup
enumerator kINPUTMUX_Gpio2pineventtrig0ToT1swkup
enumerator kINPUTMUX_Gpio3pineventtrig0ToT1swkup
enumerator kINPUTMUX_Gpio4pineventtrig0ToT1swkup
enumerator kINPUTMUX_Lpspi0receiveddatawordToT1swkup

enumerator kINPUTMUX_Lpspi1receiveddatawordToT1swkup
enumerator kINPUTMUX_Lpuart0receiveddatawordToT1swkup
enumerator kINPUTMUX_Lpuart1receiveddatawordToT1swkup
enumerator kINPUTMUX_Lpuart2receiveddatawordToT1swkup
enumerator kINPUTMUX_Lpuart3receiveddatawordToT1swkup
enumerator kINPUTMUX_Lpuart4receiveddatawordToT1swkup
enumerator kINPUTMUX_Adc1tcomp0ToT1swkup
enumerator kINPUTMUX_Adc1tcomp1ToT1swkup
enumerator kINPUTMUX_Adc1tcomp2ToT1swkup
enumerator kINPUTMUX_Adc1tcomp3ToT1swkup
enumerator kINPUTMUX_Ctimer3mat2ToT1swkup
enumerator kINPUTMUX_Ctimer3mat3ToT1swkup
enumerator kINPUTMUX_Ctimer4mat2ToT1swkup
enumerator kINPUTMUX_Ctimer4mat3ToT1swkup
enumerator kINPUTMUX_Lpspi2receiveddatawordToT1swkup
enumerator kINPUTMUX_Lpspi3receiveddatawordToT1swkup
enumerator kINPUTMUX_Gpio0pineventtrig1ToT1swkup
enumerator kINPUTMUX_Gpio1pineventtrig1ToT1swkup
enumerator kINPUTMUX_Gpio2pineventtrig1ToT1swkup
enumerator kINPUTMUX_Gpio3pineventtrig1ToT1swkup
enumerator kINPUTMUX_Gpio4pineventtrig1ToT1swkup
enumerator kINPUTMUX_ArmtxeVToDac1trig
enumerator kINPUTMUX_Aoi0out0ToDac1trig
enumerator kINPUTMUX_Aoi0out1ToDac1trig
enumerator kINPUTMUX_Aoi0out2ToDac1trig
enumerator kINPUTMUX_Aoi0out3ToDac1trig
enumerator kINPUTMUX_Cmp0outToDac1trig
enumerator kINPUTMUX_Ctimer0mat0ToDac1trig
enumerator kINPUTMUX_Ctimer0mat1ToDac1trig
enumerator kINPUTMUX_Ctimer1mat0ToDac1trig
enumerator kINPUTMUX_Ctimer1mat1ToDac1trig
enumerator kINPUTMUX_Ctimer2mat0ToDac1trig
enumerator kINPUTMUX_Ctimer2mat1ToDac1trig

enumerator kINPUTMUX_Lptmr0ToDac1trig
enumerator kINPUTMUX_Gpio0pineventtrig0ToDac1trig
enumerator kINPUTMUX_Gpio1pineventtrig0ToDac1trig
enumerator kINPUTMUX_Gpio2pineventtrig0ToDac1trig
enumerator kINPUTMUX_Gpio3pineventtrig0ToDac1trig
enumerator kINPUTMUX_Gpio4pineventtrig0ToDac1trig
enumerator kINPUTMUX_WuuToDac1trig
enumerator kINPUTMUX_Adc0tcomp0ToDac1trig
enumerator kINPUTMUX_Adc0tcomp1ToDac1trig
enumerator kINPUTMUX_Adc1tcomp0ToDac1trig
enumerator kINPUTMUX_Adc1tcomp1ToDac1trig
enumerator kINPUTMUX_Ctimer3mat0ToDac1trig
enumerator kINPUTMUX_Ctimer3mat1ToDac1trig
enumerator kINPUTMUX_Ctimer4mat0ToDac1trig
enumerator kINPUTMUX_Ctimer4mat1ToDac1trig
enumerator kINPUTMUX_Gpio0pineventtrig1ToDac1trig
enumerator kINPUTMUX_Gpio1pineventtrig1ToDac1trig
enumerator kINPUTMUX_Gpio2pineventtrig1ToDac1trig
enumerator kINPUTMUX_Gpio3pineventtrig1ToDac1trig
enumerator kINPUTMUX_Gpio4pineventtrig1ToDac1trig
enumerator kINPUTMUX_Aoi0out0ToTrigout
enumerator kINPUTMUX_Aoi0out1ToTrigout
enumerator kINPUTMUX_Aoi0out2ToTrigout
enumerator kINPUTMUX_Aoi0out3ToTrigout
enumerator kINPUTMUX_Cmp0outToTrigout
enumerator kINPUTMUX_Lpuart0ipppdolpuarttxdToTrigout
enumerator kINPUTMUX_Lpuart1ipppdolpuarttxdToTrigout
enumerator kINPUTMUX_Lpuart2ipppdolpuarttxdToTrigout
enumerator kINPUTMUX_Lpuart3ipppdolpuarttxdToTrigout
enumerator kINPUTMUX_Lpuart4ipppdolpuarttxdToTrigout
enumerator kINPUTMUX_Lpuart5ipppdolpuarttxdToTrigout
enumerator kINPUTMUX_Usb1startofframeToTrigout
enumerator kINPUTMUX_EnetppsoutputToTrigout

enumerator kINPUTMUX_Gpiop016ToSmartdmatrix
enumerator kINPUTMUX_Gpiop017ToSmartdmatrix
enumerator kINPUTMUX_Gpiop18ToSmartdmatrix
enumerator kINPUTMUX_Gpiop19ToSmartdmatrix
enumerator kINPUTMUX_Gpiop110ToSmartdmatrix
enumerator kINPUTMUX_Gpiop111ToSmartdmatrix
enumerator kINPUTMUX_Gpiop112ToSmartdmatrix
enumerator kINPUTMUX_Gpiop113ToSmartdmatrix
enumerator kINPUTMUX_Gpiop20ToSmartdmatrix
enumerator kINPUTMUX_Gpiop21ToSmartdmatrix
enumerator kINPUTMUX_Gpiop22ToSmartdmatrix
enumerator kINPUTMUX_Gpiop23ToSmartdmatrix
enumerator kINPUTMUX_Gpiop26ToSmartdmatrix
enumerator kINPUTMUX_Gpiop38ToSmartdmatrix
enumerator kINPUTMUX_Gpiop39ToSmartdmatrix
enumerator kINPUTMUX_Gpiop310ToSmartdmatrix
enumerator kINPUTMUX_Gpiop311ToSmartdmatrix
enumerator kINPUTMUX_Gpiop312ToSmartdmatrix
enumerator kINPUTMUX_Gpio0pineventtrig0ToSmartdmatrix
enumerator kINPUTMUX_Gpio1pineventtrig0ToSmartdmatrix
enumerator kINPUTMUX_Gpio2pineventtrig0ToSmartdmatrix
enumerator kINPUTMUX_Gpio3pineventtrig0ToSmartdmatrix
enumerator kINPUTMUX_Gpio4pineventtrig0ToSmartdmatrix
enumerator kINPUTMUX_ArmtxevToSmartdmatrix
enumerator kINPUTMUX_Aoi0out0ToSmartdmatrix
enumerator kINPUTMUX_Dma0irqToSmartdmatrix
enumerator kINPUTMUX_WuirqToSmartdmatrix
enumerator kINPUTMUX_Ctimer0mat2ToSmartdmatrix
enumerator kINPUTMUX_Ctimer0mat3ToSmartdmatrix
enumerator kINPUTMUX_Ctimer1mat2ToSmartdmatrix
enumerator kINPUTMUX_Ctimer1mat3ToSmartdmatrix
enumerator kINPUTMUX_Ctimer2mat2ToSmartdmatrix
enumerator kINPUTMUX_Ctimer2mat3ToSmartdmatrix

enumerator kINPUTMUX__Ctimer3mat2ToSmartdmatrix
enumerator kINPUTMUX__Ctimer3mat3ToSmartdmatrix
enumerator kINPUTMUX__Ctimer4mat2ToSmartdmatrix
enumerator kINPUTMUX__Ctimer4mat3ToSmartdmatrix
enumerator kINPUTMUX__OstimerirqToSmartdmatrix
enumerator kINPUTMUX__RtcirqToSmartdmatrix
enumerator kINPUTMUX__UtickirqToSmartdmatrix
enumerator kINPUTMUX__WdtirqToSmartdmatrix
enumerator kINPUTMUX__WakeuptimerirqToSmartdmatrix
enumerator kINPUTMUX__Can0irqToSmartdmatrix
enumerator kINPUTMUX__Can1irqToSmartdmatrix
enumerator kINPUTMUX__Flexio0irqToSmartdmatrix
enumerator kINPUTMUX__Flexio0shifer0dmareqToSmartdmatrix
enumerator kINPUTMUX__Flexio0shifer1dmareqToSmartdmatrix
enumerator kINPUTMUX__Flexio0shifer2dmareqToSmartdmatrix
enumerator kINPUTMUX__Flexio0shifer3dmareqToSmartdmatrix
enumerator kINPUTMUX__I3c0irqToSmartdmatrix
enumerator kINPUTMUX__Lpi2c0irqToSmartdmatrix
enumerator kINPUTMUX__Lpi2c1irqToSmartdmatrix
enumerator kINPUTMUX__Lpspi0irqToSmartdmatrix
enumerator kINPUTMUX__Lpspi1irqToSmartdmatrix
enumerator kINPUTMUX__Lpuart0irqToSmartdmatrix
enumerator kINPUTMUX__Lpuart1irqToSmartdmatrix
enumerator kINPUTMUX__Lpuart2irqToSmartdmatrix
enumerator kINPUTMUX__Lpuart3irqToSmartdmatrix
enumerator kINPUTMUX__Usb1startofframeToSmartdmatrix
enumerator kINPUTMUX__Adc0irqToSmartdmatrix
enumerator kINPUTMUX__Adc1irqToSmartdmatrix
enumerator kINPUTMUX__Cmp0irqToSmartdmatrix
enumerator kINPUTMUX__Cmp0outToSmartdmatrix
enumerator kINPUTMUX__Dac0irqToSmartdmatrix
enumerator kINPUTMUX__Dma1irqToSmartdmatrix
enumerator kINPUTMUX__Dac1irqToSmartdmatrix

```
enumerator kINPUTMUX__Tsi0endofscanirqToSmartdmatrixg
enumerator kINPUTMUX__Tsi0outofrangeirqToSmartdmatrixg
enumerator kINPUTMUX__EnetqosirqToSmartdmatrixg
enumerator kINPUTMUX__Tenbaset1sirqToSmartdmatrixg
enumerator kINPUTMUX__ErmininterruptToSmartdmatrixg
enumerator kINPUTMUX__Tmprout0ToSmartdmatrixg
enumerator kINPUTMUX__Tmprout1ToSmartdmatrixg
```

```
typedef enum _inputmux_index_t inputmux_index_t
```

```
typedef enum _inputmux_connection_t inputmux_connection_t
    INPUTMUX connections type.
```

```
CTIMER0CAP_REG
```

Periphinmux IDs.

```
TIMER0TRIG_REG
```

```
CTIMER1CAP_REG
```

```
TIMER1TRIG_REG
```

```
CTIMER2CAP_REG
```

```
TIMER2TRIG_REG
```

```
SMARTDMA_TRIG_REG
```

```
LPSPI2_TRIG_REG
```

```
LPSPI3_TRIG_REG
```

```
LPSPI4_TRIG_REG
```

```
LPSPI5_TRIG_REG
```

```
ENET_TRIG_IN_REG
```

```
T1S_WKUP_REG
```

```
FREQMEAS_REF_REG
```

```
FREQMEAS_TAR_REG
```

```
CTIMER3CAP_REG
```

```
TIMER3TRIG_REG
```

```
CTIMER4CAP_REG
```

```
TIMER4TRIG_REG
```

```
CMP0_TRIG_REG
```

```
ADC0_TRIG_REG
```

```
ADC1_TRIG_REG
```

```
DAC0_TRIG_REG
```

DAC1_TRIG_REG

AOI0_INPUT_REG

TSI0_TRIG_INPUT_REG

TRIG_OUT_REG

LPI2C2_TRIG_REG

LPI2C3_TRIG_REG

LPI2C4_TRIG_REG

LPI2C0_TRIG_REG

LPI2C1_TRIG_REG

LPSPi0_TRIG_REG

LPSPi1_TRIG_REG

LPUART0_REG

LPUART1_REG

LPUART2_REG

LPUART3_REG

LPUART4_REG

LPUART5_REG

FLEXIO_TRIG_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. –

2.41 KB Driver

`enum kb_operation_t`

Details of the operation to be performed by the ROM.

The `kRomAuthenticateImage` operation requires the entire signed image to be available to the application.

Values:

enumerator `kRomAuthenticateImage`

Authenticate a signed image.

enumerator `kRomLoadImage`

Load SB file.

enumerator `kRomOperationCount`

Values:

enumerator `kStatus_RomLdrDataUnderrun`

API needs to be called again with more data(next data chunk)

enumerator `kStatus_RomLdrJumpReturned`

API finished its execution but the jump to user code returned

enumerator `kStatus_RomLdrRollbackBlocked`

New firmware version is lesser than the present one

enumerator `kStatus_RomLdrPendingJumpCommand`

Returned by `kb_execute`; Call of `kb_finish` is needed to do the requested jump

enumerator `kStatus_RomApiBufferSizeNotEnough`

API cannot proceed because the provided buffer(via call of `kb_init` in options argument) is not large enough

enumerator `kStatus_RomApiInvalidBuffer`

API cannot proceed because the provided buffer pointer is NULL or buffer size is 0

`typedef struct kb_buffer_desc_tag kb_buffer_desc_t`


```
typedef struct kb_opaque_session_ref_tag kb_session_ref_t
```

```
status_t KB_Init(kb_session_ref_t **session, const kb_options_t *options)
```

This API is used to initialize bootloader and nboot context necessary to process sb4 file format.

Parameters

- *session* – A double pointer to a *kb_session_ref_t* object where the initialized session reference will be stored.
- *options* – A pointer to a *kb_options_t* structure containing configuration settings for initializing the session.

```
status_t KB_Deinit(kb_session_ref_t *session)
```

This API is used to release nboot context and finalize sb4 file processing.

Parameters

- *session* – A pointer to an existing *kb_session_ref_t* session object that should be released and deinitialized.

```
status_t KB_Execute(kb_session_ref_t *session, const uint8_t *data, uint32_t dataLength)
```

This API is used to decrypt sb4 file and store signed image contents specified by loader command supported while generating sb4 image through json configuration.

Parameters

- *session* – Session pointer obtained from *kb_init* function.
- *data* – Pointer to sb file data..
- *dataLength* – sb file data length in bytes.

```
FSL_KBAPI_DRIVER_VERSION
```

KBAPI driver version 2.0.0.

```
uint32_t address
```

```
uint32_t length
```

```
uint32_t profile
```

```
uint32_t minBuildNumber
```

```
uint32_t overrideSBBootSectionID
```

```
uint32_t *userSBKEK
```

```
uint32_t regionCount
```

```
const kb_region_t *regions
```

```
uint32_t profile
```

```
uint32_t minBuildNumber
```

```
uint32_t maxImageLength
```

```
uint32_t *userRHK
```

```
uint32_t version
```

Should be set to *kKbootApiVersion*.

```
uint8_t *buffer
```

Caller-provided buffer used by Kboot.

uint32_t bufferLength

kb_operation_t op

kb_authenticate_t authenticate
Settings for kb_authenticate_t operation.

kb_load_sb_t loadSB
Settings for kb_load_sb_t operation.

union kb_options_t

uint8_t *buf

uint32_t len

uint32_t allocated

kb_options_t options

kb_buffer_desc_t buffer_desc

void *op_context

struct kb_region_t
#include <fsl_kbapi.h> Memory region definition.

struct kb_load_sb_t
#include <fsl_kbapi.h>

struct kb_authenticate_t
#include <fsl_kbapi.h>

struct kb_options_t
#include <fsl_kbapi.h>

struct kb_buffer_desc_tag
#include <fsl_kbapi.h>

struct kb_opaque_session_ref_tag
#include <fsl_kbapi.h>

union __unnamed31__

Public Members

kb_authenticate_t authenticate
Settings for kb_authenticate_t operation.

kb_load_sb_t loadSB
Settings for kb_load_sb_t operation.

2.42 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 USB CDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM

Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART

Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART

Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_UART

Debug console based on LPC_UART.

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 specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)

Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true**, else return **false**.

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value (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 I2C status codes.

enumerator kStatusGroup_LPSCI

Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART

Group number for LPUART status codes.

enumerator kStatusGroup_SPI

Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPIDSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator `kStatusGroup_SMARTCARD`
Group number for SMARTCARD status codes.

enumerator `kStatusGroup_LMEM`
Group number for LMEM status codes.

enumerator `kStatusGroup_QSPI`
Group number for QSPI status codes.

enumerator `kStatusGroup_DMA`
Group number for DMA status codes.

enumerator `kStatusGroup_EDMA`
Group number for EDMA status codes.

enumerator `kStatusGroup_DMAMGR`
Group number for DMAMGR status codes.

enumerator `kStatusGroup_FLEXCAN`
Group number for FlexCAN status codes.

enumerator `kStatusGroup_LTC`
Group number for LTC status codes.

enumerator `kStatusGroup_FLEXIO_CAMERA`
Group number for FLEXIO CAMERA status codes.

enumerator `kStatusGroup_LPC_SPI`
Group number for LPC_SPI status codes.

enumerator `kStatusGroup_LPC_USART`
Group number for LPC_USART status codes.

enumerator `kStatusGroup_DMIC`
Group number for DMIC status codes.

enumerator `kStatusGroup_SDIF`
Group number for SDIF status codes.

enumerator `kStatusGroup_SPIFI`
Group number for SPIFI status codes.

enumerator `kStatusGroup_OTP`
Group number for OTP status codes.

enumerator `kStatusGroup_MCAN`
Group number for MCAN status codes.

enumerator `kStatusGroup_CAAM`
Group number for CAAM status codes.

enumerator `kStatusGroup_ECSPI`
Group number for ECSPI status codes.

enumerator `kStatusGroup_USDHC`
Group number for USDHC status codes.

enumerator `kStatusGroup_LPC_I2C`
Group number for LPC_I2C status codes.

enumerator `kStatusGroup_DCP`
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALIZEDMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator kStatusGroup_SDIO SLV
Group number for SDIO SLV status codes.

enumerator kStatusGroup_MECC
Group number for MECC status codes.

enumerator kStatusGroup_ENET_QOS
Group number for ENET_QOS status codes.

enumerator kStatusGroup_LOG
Group number for LOG status codes.

enumerator kStatusGroup_I3CBUS
Group number for I3CBUS status codes.

enumerator kStatusGroup_QSCI
Group number for QSCI status codes.

enumerator kStatusGroup_ELEMU
Group number for ELEMU status codes.

enumerator kStatusGroup_QUEUEDSPI
Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.

enumerator kStatusGroup_CLIF
Group number for CLIF status codes.

enumerator kStatusGroup_BMA
Group number for BMA status codes.

enumerator kStatusGroup_NETC
Group number for NETC status codes.

enumerator kStatusGroup_ELE
Group number for ELE status codes.

enumerator kStatusGroup_GLIKEY
Group number for GLIKEY status codes.

enumerator kStatusGroup_AON_POWER
Group number for AON_POWER status codes.

enumerator kStatusGroup_AON_COMMON
Group number for AON_COMMON status codes.

enumerator kStatusGroup_ENDAT3
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

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.

2.43 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.

2.44 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.

2.45 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.

2.46 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.

2.47 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.

2.48 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.

2.49 LPSPI: Low Power Serial Peripheral Interface

2.50 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.

2.51 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

2.52 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

2.53 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.54 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 __unnamed61__`

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 __unnamed63__`

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 __unnamed65__`

Public Members

`uint8_t *volatile rxData`
Address of remaining data to receive.

`uint16_t *volatile rxData16`
Address of remaining data to receive.

`union __unnamed67__`

Public Members

`uint8_t *rxRingBuffer`
Start address of the receiver ring buffer.

`uint16_t *rxRingBuffer16`
Start address of the receiver ring buffer.

2.55 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

2.56 ENET: Ethernet Driver

2.57 Mcx_enet

Defines the status return codes for transaction.

Values:

enumerator kStatus_ENET_InitMemoryFail

Status code 4000. Init failed since buffer memory was not enough.

enumerator kStatus_ENET_RxFrameError

Status code 4001. A frame received but data error occurred.

enumerator kStatus_ENET_RxFrameFail

Status code 4002. Failed to receive a frame.

enumerator kStatus_ENET_RxFrameEmpty

Status code 4003. No frame arrived.

enumerator kStatus_ENET_RxFrameDrop

Status code 4004. Rx frame was dropped since there's no buffer memory.

enumerator kStatus_ENET_TxFrameBusy

Status code 4005. There were no resources for Tx operation.

enumerator kStatus_ENET_TxFrameFail

Status code 4006. Transmit frame failed.

enumerator kStatus_ENET_TxFrameOverLen

Status code 4007. Failed to send an oversize frame.

enum _enet_mii_mode

Defines the MII/RMII mode for data interface between the MAC and the PHY.

Values:

enumerator kENET_MiiMode

MII mode for data interface.

enumerator kENET_RmiiMode

RMII mode for data interface.

enum _enet_mii_speed

Defines the 10/100 Mbps speed for the MII data interface.

Values:

enumerator kENET_MiiSpeed10M

Speed 10 Mbps.

enumerator kENET_MiiSpeed100M

Speed 100 Mbps.

enum _enet_mii_duplex

Defines the half or full duplex for the MII data interface.

Values:

enumerator kENET_MiiHalfDuplex

Half duplex mode.

enumerator kENET_MiiFullDuplex

Full duplex mode.

enum _enet_dma_burstlen

Define the DMA maximum transmit burst length.

Values:

enumerator kENET_BurstLen1

DMA burst length 1.

enumerator kENET_BurstLen2

DMA burst length 2.

enumerator kENET_BurstLen4

DMA burst length 4.

enumerator kENET_BurstLen8

DMA burst length 8.

enumerator kENET_BurstLen16

DMA burst length 16.

enumerator kENET_BurstLen32

DMA burst length 32.

enumerator kENET_BurstLen64

DMA burst length 64. eight times enabled.

enumerator kENET_BurstLen128

DMA burst length 128. eight times enabled.

enumerator kENET_BurstLen256

DMA burst length 256. eight times enabled.

enum _enet_desc_flag

Define the flag for the descriptor.

Values:

enumerator kENET_MiddleFlag

It's a middle descriptor of the frame.

enumerator kENET_LastFlagOnly

It's the last descriptor of the frame.

enumerator kENET_FirstFlagOnly

It's the first descriptor of the frame.

enumerator kENET_FirstLastFlag

It's the first and last descriptor of the frame.

enum _enet_systime_op

Define the system time adjust operation control.

Values:

enumerator kENET_SystimeAdd

System time add to.

enumerator kENET_SystimeSubtract

System time subtract.

enum _enet_ts_rollover_type

Define the system time rollover control.

Values:

enumerator kENET_BinaryRollover

System time binary rollover.

enumerator kENET_DigitalRollover

System time digital rollover.

enum _enet_special_config

Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, there is no need to set this control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the enet_config_t.

Note: “kENET_StoreAndForward” is recommended to be set when the ENET_PTP1588FEATURE_REQUIRED is defined or else the timestamp will be mess-up when the overflow happens.

Values:

enumerator kENET_DescDoubleBuffer

The double buffer is used in the Tx/Rx descriptor.

enumerator kENET_StoreAndForward

The Rx/Tx store and forward enable.

enumerator kENET_PromiscuousEnable

The promiscuous enabled.

enumerator kENET_FlowControlEnable

The flow control enabled.

enumerator kENET_BroadCastRxDisable

The broadcast disabled.

enumerator kENET_MulticastAllEnable

All multicast are passed.

enumerator kENET_8023AS2KPacket

8023as support for 2K packets.

enumerator kENET_RxChecksumOffloadEnable

The Rx checksum offload enabled.

enum _enet_dma_interrupt_enable

List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

Values:

enumerator kENET_DmaTx

Tx interrupt.

enumerator kENET_DmaTxStop

Tx stop interrupt.

enumerator kENET_DmaTxBuffUnavail

Tx buffer unavailable.

enumerator kENET_DmaRx

Rx interrupt.

enumerator kENET_DmaRxBuffUnavail

Rx buffer unavailable.

enumerator kENET_DmaRxStop

Rx stop.

enumerator kENET_DmaRxWatchdogTimeout

Rx watchdog timeout.

enumerator kENET_DmaEarlyTx

Early transmit.

enumerator kENET_DmaEarlyRx

Early receive.

enumerator kENET_DmaBusErr

Fatal bus error.

enum _enet_mac_interrupt_enable

List of mac interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

Values:

enumerator kENET_MacPmt

enumerator kENET_MacTimestamp

enum _enet_event

Defines the common interrupt event for callback use.

Values:

enumerator kENET_RxIntEvent

Receive interrupt event.

enumerator kENET_TxIntEvent

Transmit interrupt event.

enumerator kENET_WakeUpIntEvent

Wake up interrupt event.

enumerator kENET_TimeStampIntEvent

Time stamp interrupt event.

enum _enet_dma_tx_sche

Define the DMA transmit arbitration for multi-queue.

Values:

enumerator kENET_FixPri

Fixed priority. channel 0 has lower priority than channel 1.

enumerator kENET_WeightStrPri

Weighted(burst length) strict priority.

enumerator kENET_WeightRoundRobin

Weighted (weight factor) round robin.

enum _enet_mtl_multiqueue_txsche

Define the MTL Tx scheduling algorithm for multiple queues/rings.

Values:

enumerator kENET_txWeightRR

Tx weight round-robin.

enumerator kENET_txStrPrio

Tx strict priority.

enum _enet_mtl_multiqueue_rxsche

Define the MTL Rx scheduling algorithm for multiple queues/rings.

Values:

enumerator kENET_rxStrPrio

Tx weight round-robin, Rx strict priority.

enumerator kENET_rxWeightStrPrio

Tx strict priority, Rx weight strict priority.

enum _enet_mtl_rxqueuemap

Define the MTL Rx queue and DMA channel mapping.

Values:

enumerator kENET_StaticDirectMap

The received frame in Rx Qn(n = 0,1) directly map to dma channel n.

enumerator kENET_DynamicMap

The received frame in Rx Qn(n = 0,1) map to the dma channel m(m = 0,1) related with the same Mac.

enum _enet_ptp_event_type

Defines the ENET PTP message related constant.

Values:

enumerator kENET_PtpEventMsgType

PTP event message type.

enumerator kENET_PtpSrcPortIdLen

PTP message sequence id length.

enumerator kENET_PtpEventPort

PTP event port number.

enumerator kENET_PtpGnrlPort

PTP general port number.

enum _enet_tx_offload

Define the Tx checksum offload options.

Values:

enumerator kENET_TxOffloadDisable

Disable Tx checksum offload.

enumerator kENET_TxOffloadIPHeader

Enable IP header checksum calculation and insertion.

enumerator kENET_TxOffloadIPHeaderPlusPayload

Enable IP header and payload checksum calculation and insertion.

enumerator kENET_TxOffloadAll

Enable IP header, payload and pseudo header checksum calculation and insertion.

enum _enet_vlan_tpid

Ethernet VLAN Tag protocol identifiers.

Values:

enumerator kENET_StanCvlan

C-VLAN 0x8100.

enumerator kENET_StanSvlan

S-VLAN 0x88A8.

enum _enet_vlan_ops

Ethernet VLAN operations.

Values:

enumerator kENET_NoOps

Not do anything.

enumerator kENET_VlanRemove

Remove VLAN Tag.

enumerator kENET_VlanInsert

Insert VLAN Tag.

enumerator kENET_VlanReplace

Replace VLAN Tag.

enum _enet_vlan_strip

Ethernet VLAN strip setting.

Values:

enumerator kENET_VlanNotStrip

Not strip frame.

enumerator kENET_VlanFilterPassStrip

Strip if VLAN filter passes.

enumerator kENET_VlanFilterFailStrip

Strip if VLAN filter fails.

enumerator kENET_VlanAlwaysStrip

Always strip.

enum _enet_vlan_tx_channel

Ethernet VLAN Tx channels.

Values:

enumerator kENET_VlanTagAllChannels

VLAN tag is inserted for every packets transmitted by the MAC.

enumerator kENET_VlanTagChannel0

VLAN tag is inserted for the frames transmitted by channel 0.

enumerator kENET_VlanTagChannel1

VLAN tag is inserted for the frames transmitted by channel 1.

typedef enum _enet_mii_mode enet_mii_mode_t

Defines the MII/RMII mode for data interface between the MAC and the PHY.

`typedef enum _enet_mii_speed enet_mii_speed_t`

Defines the 10/100 Mbps speed for the MII data interface.

`typedef enum _enet_mii_duplex enet_mii_duplex_t`

Defines the half or full duplex for the MII data interface.

`typedef enum _enet_dma_burstlen enet_dma_burstlen_t`

Define the DMA maximum transmit burst length.

`typedef enum _enet_desc_flag enet_desc_flag_t`

Define the flag for the descriptor.

`typedef enum _enet_systime_op enet_systime_op_t`

Define the system time adjust operation control.

`typedef enum _enet_ts_rollover_type enet_ts_rollover_type_t`

Define the system time rollover control.

`typedef enum _enet_special_config enet_special_config_t`

Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, there is no need to set this control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the enet_config_t.

Note: “kENET_StoreAndForward” is recommended to be set when the ENET_PTP1588FEATURE_REQUIRED is defined or else the timestamp will be mess-up when the overflow happens.

`typedef enum _enet_dma_interrupt_enable enet_dma_interrupt_enable_t`

List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

`typedef enum _enet_mac_interrupt_enable enet_mac_interrupt_enable_t`

List of mac interrupts supported by the ENET interrupt. This enumeration uses one-hot encoding to allow a logical OR of multiple members.

`typedef enum _enet_event enet_event_t`

Defines the common interrupt event for callback use.

`typedef enum _enet_dma_tx_sche enet_dma_tx_sche_t`

Define the DMA transmit arbitration for multi-queue.

`typedef enum _enet_mtl_multiqueue_txsche enet_mtl_multiqueue_txsche_t`

Define the MTL Tx scheduling algorithm for multiple queues/rings.

`typedef enum _enet_mtl_multiqueue_rxsche enet_mtl_multiqueue_rxsche_t`

Define the MTL Rx scheduling algorithm for multiple queues/rings.

`typedef enum _enet_mtl_rxqueuemap enet_mtl_rxqueuemap_t`

Define the MTL Rx queue and DMA channel mapping.

`typedef enum _enet_ptp_event_type enet_ptp_event_type_t`

Defines the ENET PTP message related constant.

`typedef enum _enet_tx_offload enet_tx_offload_t`

Define the Tx checksum offload options.

`typedef enum _enet_vlan_tpid enet_vlan_tpid_t`

Ethernet VLAN Tag protocol identifiers.

typedef enum *_enet_vlan_ops* enet_vlan_ops_t

Ethernet VLAN operations.

typedef enum *_enet_vlan_strip* enet_vlan_strip_t

Ethernet VLAN strip setting.

typedef enum *_enet_vlan_tx_channel* enet_vlan_tx_channel_t

Ethernet VLAN Tx channels.

typedef struct *_enet_rx_bd_struct* enet_rx_bd_struct_t

Defines the receive descriptor structure. It has the read-format and write-back format structures. They both have the same size with different region definition. So we define common name as the receive descriptor structure. When initialize the buffer descriptors, read-format region mask bits should be used. When Rx frame has been in the buffer descriptors, write-back format region store the Rx result information.

typedef struct *_enet_tx_bd_struct* enet_tx_bd_struct_t

Defines the transmit descriptor structure. It has the read-format and write-back format structure. They both have the same size with different region definition. So we define common name as the transmit descriptor structure. When initialize the buffer descriptors for Tx, read-format region mask bits should be used. When frame has been transmitted, write-back format region store the Tx result information.

typedef struct *_enet_tx_bd_config_struct* enet_tx_bd_config_struct_t

Defines the Tx BD configuration structure.

typedef struct *_enet_ptp_time* enet_ptp_time_t

Defines the ENET PTP time stamp structure.

typedef struct *enet_tx_reclaim_info* enet_tx_reclaim_info_t

Defines the Tx reclaim information structure.

typedef struct *_enet_tx_dirty_ring* enet_tx_dirty_ring_t

Defines the ENET transmit dirty addresses ring/queue structure.

typedef struct *_enet_buffer_config* enet_buffer_config_t

Defines the buffer descriptor configure structure.

Notes:

- The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.
- The recommended minimum Tx/Rx ring length is 4.
- The Tx/Rx descriptor tail address shall be the address pointer to the address just after the end of the last descriptor. Because only the descriptors between the start address and the tail address will be used by DMA.
- The descriptor address is the start address of all used contiguous memory. For example, the `rxDescStartAddrAlign` is the start address of `rxRingLen` contiguous descriptor memory for Rx descriptor ring 0.
- The “`*rxBufferStartAddr`” is the first element of `rxRingLen` (`2*rxRingLen` for double buffers) Rx buffers. It means the `*rxBufferStartAddr` is the Rx buffer for the first descriptor, the `*rxBufferStartAddr + 1` is the Rx buffer for the second descriptor or the Rx buffer for the second buffer in the first descriptor. So please make sure the `rxBufferStartAddr` is the address of a `rxRingLen` or `2*rxRingLen` array.

typedef struct *enet_multiqueue_config* enet_multiqueue_config_t

Defines the configuration when multi-queue is used.

```
typedef void (*enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t channel)
```

Defines the Rx memory buffer alloc function pointer.

```
typedef void (*enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t channel)
```

Defines the Rx memory buffer free function pointer.

```
typedef struct _enet_config enet_config_t
```

Defines the basic configuration structure for the ENET device.

Note:

- a. Default the signal queue is used so the “multiqueueCfg” is set default with NULL. Set the pointer with a valid configuration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

```
typedef struct _enet_handle enet_handle_t
```

```
typedef void (*enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, uint8_t channel, enet_tx_reclaim_info_t *txReclaimInfo, void *userData)
```

ENET callback function.

```
typedef struct _enet_tx_bd_ring enet_tx_bd_ring_t
```

Defines the ENET transmit buffer descriptor ring/queue structure.

```
typedef struct _enet_rx_bd_ring enet_rx_bd_ring_t
```

Defines the ENET receive buffer descriptor ring/queue structure.

```
typedef struct _enet_buffer_struct enet_buffer_struct_t
```

```
typedef struct _enet_rx_frame_attribute_struct enet_rx_frame_attribute_t
```

Rx frame attribute structure.

```
typedef struct _enet_rx_frame_error enet_rx_frame_error_t
```

Defines the Rx frame error structure.

```
typedef struct _enet_rx_frame_struct enet_rx_frame_struct_t
```

Defines the Rx frame data structure.

```
typedef struct _enet_tx_config_struct enet_tx_config_struct_t
```

```
typedef struct _enet_tx_frame_struct enet_tx_frame_struct_t
```

```
typedef struct _enet_vlan_tag enet_vlan_tag_t
```

Ethernet VLAN Tag.

```
typedef struct _enet_vlan_tx_config enet_vlan_tx_config_t
```

Ethernet VLAN configuration for Tx.

```
typedef struct _enet_vlan_ctrl enet_vlan_ctrl_t
```

Ethernet VLAN control.

```
typedef void (*enet_isr_t)(ENET_Type *base, enet_handle_t *handle)
```

```
const clock_ip_name_t s_enetClock[]
```

Pointers to enet clocks for each instance.

```
ENET_FRAME_MAX_FRAMELEN
```

Default maximum ethernet frame size.

```
ENET_FCS_LEN
```

Ethernet Rx frame FCS length.

ENET_ADDR_ALIGNMENT

Recommended ethernet buffer alignment.

ENET_BUFF_ALIGNMENT

Receive buffer alignment shall be 4bytes-aligned.

ENET_RING_NUM_MAX

The maximum number of Tx/Rx descriptor rings.

ENET_MTL_RXFIFOSIZE

The Rx fifo size.

ENET_MTL_TXFIFOSIZE

The Tx fifo size.

ENET_MACINT_ENUM_OFFSET

The offset for mac interrupt in enum type.

ENET_FRAME_TX_LEN_LIMITATION

The Tx frame length software limitation.

ENET_FRAME_RX_ERROR_BITS(x)

The Rx frame error bits field.

void ENET_GetDefaultConfig(*enet_config_t* *config)

Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET configure structure for ENET_Init(). User may use the initialized structure unchanged in ENET_Init(), or modify some fields of the structure before calling ENET_Init(). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

- config – The ENET mac controller configuration structure pointer.

void ENET_Init(ENET_Type *base, const *enet_config_t* *config, uint8_t *macAddr, uint32_t clkSrcHz)

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET basic configuration.

Note: As our transactional transmit API use the zero-copy transmit buffer. So there are two thing we emphasize here:

- a. Tx buffer free/requeue for application should be done in the Tx interrupt handler. Please set callback: kENET_TxIntEvent with Tx buffer free/requeue process APIs.
 - b. The Tx interrupt is forced to open.
-

Parameters

- base – ENET peripheral base address.
- config – ENET mac configuration structure pointer. The “enet_config_t” type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.

- `clkSrcHz` – ENET input reference clock.

`void ENET_Deinit(ENET_Type *base)`

Deinitializes the ENET module.

This function gates the module clock and disables the ENET module.

Parameters

- `base` – ENET peripheral base address.

`status_t ENET_DescriptorInit(ENET_Type *base, enet_config_t *config, enet_buffer_config_t *bufferConfig)`

Initialize for all ENET descriptors.

Note: This function finishes all Tx/Rx descriptors initialization. The descriptor initialization should be called after `ENET_Init()`.

Parameters

- `base` – ENET peripheral base address.
- `config` – The configuration for ENET.
- `bufferConfig` – All buffers configuration.

`status_t ENET_RxBufferAllocAll(ENET_Type *base, enet_handle_t *handle)`

Allocates Rx buffers for all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will populate initial buffers in all BDs for receiving. Then `ENET_GetRxFrame()` is used to get Rx frame with zero copy, it will allocate new buffer to replace the buffer in BD taken by application, application should free those buffers after they're used.

Note: This function should be called after `ENET_CreateHandler()` and buffer allocating callback function should be ready.

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.

`void ENET_RxBufferFreeAll(ENET_Type *base, enet_handle_t *handle)`

Frees Rx buffers in all BDs. It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will free left buffers in all BDs.

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.

`void ENET_StartRxTx(ENET_Type *base, uint8_t txRingNum, uint8_t rxRingNum)`

Starts the ENET Tx/Rx. This function enable the Tx/Rx and starts the Tx/Rx DMA. This shall be set after ENET initialization and before starting to receive the data.

Note: This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

Parameters

- base – ENET peripheral base address.
- rxRingNum – The number of the used Rx rings. It shall not be larger than the ENET_RING_NUM_MAX(2). If the ringNum is set with 1, the ring 0 will be used.
- txRingNum – The number of the used Tx rings. It shall not be larger than the ENET_RING_NUM_MAX(2). If the ringNum is set with 1, the ring 0 will be used.

```
void ENET_SetISRHandler(ENET_Type *base, enet_isr_t ISRHandler)
```

Set the second level IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

```
static inline void ENET_SetMII(ENET_Type *base, enet_mii_speed_t speed, enet_mii_duplex_t duplex)
```

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and duplex for MAC.

Parameters

- base – ENET peripheral base address.
- speed – The speed of the RMII mode.
- duplex – The duplex of the RMII mode.

```
void ENET_SetSMI(ENET_Type *base, uint32_t clkSrcHz)
```

Sets the ENET SMI(serial management interface)- MII management interface.

Parameters

- base – ENET peripheral base address.
- clkSrcHz – ENET peripheral clock source.

```
static inline bool ENET_IsSMIBusy(ENET_Type *base)
```

Checks if the SMI is busy.

Parameters

- base – ENET peripheral base address.

Returns

The status of MII Busy status.

```
static inline uint16_t ENET_ReadSMIData(ENET_Type *base)
```

Reads data from the PHY register through SMI interface.

Parameters

- base – ENET peripheral base address.

Returns

The data read from PHY

```
void ENET_StartSMIWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)
```

Sends the MDIO IEEE802.3 Clause 22 format write command.

Parameters

- base – ENET peripheral base address.

- phyAddr – The PHY address.
- regAddr – The PHY register.
- data – The data written to PHY.

void ENET_StartSMIRRead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr)

Sends the MDIO IEEE802.3 Clause 22 format read command.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.

status_t ENET_MDIOWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- data – The data written to PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

status_t ENET_MDIORead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register.
- pData – The data read from PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

uint32_t ENET_GetInstance(ENET_Type *base)

Get the ENET instance from peripheral base address.

Parameters

- base – ENET peripheral base address.

Returns

ENET instance.

static inline void ENET_SetMacAddr(ENET_Type *base, uint8_t *macAddr)

Sets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.

- `macAddr` – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

`void ENET_GetMacAddr(ENET_Type *base, uint8_t *macAddr)`

Gets the ENET module Mac address.

Parameters

- `base` – ENET peripheral base address.
- `macAddr` – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

`static inline void ENET_AcceptAllMulticast(ENET_Type *base)`

Enable ENET device to accept all multicast frames.

Parameters

- `base` – ENET peripheral base address.

`static inline void ENET_RejectAllMulticast(ENET_Type *base)`

ENET device reject to accept all multicast frames.

Parameters

- `base` – ENET peripheral base address.

`void ENET_EnterPowerDown(ENET_Type *base, uint32_t *wakeFilter)`

Set the MAC to enter into power down mode. the remote power wake up frame and magic frame can wake up the ENET from the power down mode.

Parameters

- `base` – ENET peripheral base address.
- `wakeFilter` – The wakeFilter provided to configure the wake up frame filter. Set the wakeFilter to NULL is not required. But if you have the filter requirement, please make sure the wakeFilter pointer shall be eight continuous 32-bits configuration.

`static inline void ENET_ExitPowerDown(ENET_Type *base)`

Set the MAC to exit power down mode. Exit from the power down mode and recover to normal work mode.

Parameters

- `base` – ENET peripheral base address.

`status_t ENET_SetVlanCtrl(ENET_Type *base, enet_vlan_ctrl_t *control)`

Set VLAN control.

Parameters

- `base` – ENET peripheral base address.
- `control` – VLAN control configuration.

`status_t ENET_SetTxOuterVlan(ENET_Type *base, enet_vlan_tx_config_t *config, enet_vlan_tx_channel_t channel)`

Set Tx outer VLAN configuration.

Parameters

- `base` – ENET peripheral base address.
- `config` – Tx VLAN operation configuration.
- `channel` – The channel to apply this configuration.

status_t ENET_SetTxInnerVlan(ENET_Type *base, *enet_vlan_tx_config_t* *config)

Set Tx inner VLAN configuration.

Parameters

- base – ENET peripheral base address.
- config – Tx VLAN operation configuration.

void ENET_EnableInterrupts(ENET_Type *base, uint32_t mask)

Enables the ENET DMA and MAC interrupts.

This function enables the ENET interrupts according to the provided mask. Already enabled interrupts stay enabled even if not listed in the provided mask. The mask is a logical OR of *enet_dma_interrupt_enable_t* and *enet_mac_interrupt_enable_t*. For example, to enable the dma and mac interrupt, do the following.

```
ENET_EnableInterrupts(ENET, kENET_DmaRx | kENET_DmaTx | kENET_MacPmt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of both enumeration :: *enet_dma_interrupt_enable_t* and *enet_mac_interrupt_enable_t*.

void ENET_DisableInterrupts(ENET_Type *base, uint32_t mask)

Disables the ENET DMA and MAC interrupts.

This function disables the ENET interrupt according to the provided mask. The mask is a logical OR of *enet_dma_interrupt_enable_t* and *enet_mac_interrupt_enable_t*. For example, to disable the dma and mac interrupt, do the following.

```
ENET_DisableInterrupts(ENET, kENET_DmaRx | kENET_DmaTx | kENET_MacPmt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to disables. This is a logical OR of both enumeration :: *enet_dma_interrupt_enable_t* and *enet_mac_interrupt_enable_t*.

static inline uint32_t ENET_GetDmaInterruptStatus(ENET_Type *base, uint8_t channel)

Gets the ENET DMA interrupt status flag.

Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than ENET_RING_NUM_MAX.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: *enet_dma_interrupt_enable_t*.

static inline void ENET_ClearDmaInterruptStatus(ENET_Type *base, uint8_t channel, uint32_t mask)

Clear the ENET DMA interrupt status flag.

Parameters

- base – ENET peripheral base address.
- channel – The DMA Channel. Shall not be larger than ENET_RING_NUM_MAX.

- mask – The event status of the interrupt source. This is the logical OR of members of the enumeration :: `enet_dma_interrupt_enable_t`.

static inline uint32_t ENET_GetMacInterruptStatus(ENET_Type *base)

Gets the ENET MAC interrupt status flag.

Parameters

- base – ENET peripheral base address.

Returns

The event status of the interrupt source. Use the enum in `enet_mac_interrupt_enable_t` and right shift `ENET_MACINT_ENUM_OFFSET` to mask the returned value to get the exact interrupt status.

void ENET_ClearMacInterruptStatus(ENET_Type *base, uint32_t mask)

Clears the ENET mac interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the `enet_mac_interrupt_enable_t`. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_ClearMacInterruptStatus(ENET, kENET_MacPmt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration :: `enet_mac_interrupt_enable_t`.

static inline bool ENET_IsTxDescriptorDmaOwn(*enet_tx_bd_struct_t* *txDesc)

Get the Tx descriptor DMA Own flag.

Parameters

- txDesc – The given Tx descriptor.

Return values

True – the dma own Tx descriptor, false application own Tx descriptor.

void ENET_SetupTxDescriptor(*enet_tx_bd_struct_t* *txDesc, void *buffer1, uint32_t bytes1, void *buffer2, uint32_t bytes2, uint32_t framen, bool intEnable, bool tsEnable, *enet_desc_flag_t* flag, uint8_t slotNum)

Setup a given Tx descriptor. This function is a low level functional API to setup or prepare a given Tx descriptor.

Note: This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required. Transmit buffers are ‘zero-copy’ buffers, so the buffer must remain in memory until the packet has been fully transmitted. The buffers should be free or queued in the transmit interrupt irq handler.

Parameters

- txDesc – The given Tx descriptor.
- buffer1 – The first buffer address in the descriptor.
- bytes1 – The bytes in the first buffer.
- buffer2 – The second buffer address in the descriptor.
- bytes2 – The bytes in the second buffer.
- framen – The length of the frame to be transmitted.

- `intEnable` – Interrupt enable flag.
- `tsEnable` – The timestamp enable.
- `flag` – The flag of this Tx descriptor, see “enet_desc_flag_t”.
- `slotNum` – The slot num used for AV mode only.

```
static inline void ENET_UpdateTxDescriptorTail(ENET_Type *base, uint8_t channel, uint32_t txDescTailAddrAlign)
```

Update the Tx descriptor tail pointer. This function is a low level functional API to update the the Tx descriptor tail. This is called after you setup a new Tx descriptor to update the tail pointer to make the new descriptor accessible by DMA.

Parameters

- `base` – ENET peripheral base address.
- `channel` – The Tx DMA channel.
- `txDescTailAddrAlign` – The new Tx tail pointer address.

```
static inline void ENET_UpdateRxDescriptorTail(ENET_Type *base, uint8_t channel, uint32_t rxDescTailAddrAlign)
```

Update the Rx descriptor tail pointer. This function is a low level functional API to update the the Rx descriptor tail. This is called after you setup a new Rx descriptor to update the tail pointer to make the new descriptor accessible by DMA and to anouse the Rx poll command for DMA.

Parameters

- `base` – ENET peripheral base address.
- `channel` – The Rx DMA channel.
- `rxDescTailAddrAlign` – The new Rx tail pointer address.

```
static inline uint32_t ENET_GetRxDescriptor(enet_rx_bd_struct_t *rxDesc)
```

Gets the context in the ENET Rx descriptor. This function is a low level functional API to get the the status flag from a given Rx descriptor.

Note: This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required.

Parameters

- `rxDesc` – The given Rx descriptor.

Return values

The – RDES3 regions for write-back format Rx buffer descriptor.

```
void ENET_UpdateRxDescriptor(enet_rx_bd_struct_t *rxDesc, void *buffer1, void *buffer2, bool intEnable, bool doubleBuffEnable)
```

Updates the buffers and the own status for a given Rx descriptor. This function is a low level functional API to Updates the buffers and the own status for a given Rx descriptor.

Note: This must be called after all the ENET initilization. And should be called when the ENET receive/transmit is required.

Parameters

- `rxDesc` – The given Rx descriptor.

- `buffer1` – The first buffer address in the descriptor.
- `buffer2` – The second buffer address in the descriptor.
- `intEnable` – Interrupt enable flag.
- `doubleBuffEnable` – The double buffer enable flag.

```
void ENET__CreateHandler(ENET_Type *base, enet_handle_t *handle, enet_config_t *config,  
                        enet_buffer_config_t *bufferConfig, enet_callback_t callback, void  
                        *userData)
```

Create ENET Handler.

This is a transactional API and it's provided to store all datas which are needed during the whole transactional process. This API should not be used when you use functional APIs to do data Tx/Rx. This is funtion will store many data/flag for transactional use.

Parameters

- `base` – ENET peripheral base address.
- `handle` – ENET handler.
- `config` – ENET configuration.
- `bufferConfig` – ENET buffer configuration.
- `callback` – The callback function.
- `userData` – The application data.

```
status_t ENET__GetRxFramSize(ENET_Type *base, enet_handle_t *handle, uint32_t *length,  
                             uint8_t channel)
```

Gets the size of the read frame. This function gets a received frame size from the ENET buffer descriptors.

Note: The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling `ENET__GetRxFramSize`, `ENET_ReadFrame()` should be called to update the receive buffers If the result is not “`kStatus_ENET_RxFrameEmpty`”.

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler structure. This is the same handler pointer used in the `ENET_Init`.
- `length` – The length of the valid frame received.
- `channel` – The DMAC channel for the Rx.

Return values

- `kStatus_ENET_RxFrameEmpty` – No frame received. Should not call `ENET_ReadFrame` to read frame.
- `kStatus_ENET_RxFrameError` – Data error happens. `ENET_ReadFrame` should be called with NULL data and NULL length to update the receive buffers.
- `kStatus_Success` – Receive a frame Successfully then the `ENET_ReadFrame` should be called with the right data buffer and the captured data length input.


```
status_t ENET_ReadFrame(ENET_Type *base, enet_handle_t *handle, uint8_t *data, uint32_t
length, uint8_t channel, enet_ptp_time_t *timestamp)
```

Reads a frame from the ENET device. This function reads a frame from the ENET DMA descriptors. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. For example use Rx dma channel 0:

```
uint32_t length;
enet_handle_t g_handle;
Comment: Get the received frame size firstly.
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    Comment: Allocate memory here with the size of "length"
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0);
    }
    else
    {
        status = ENET_ReadFrame(ENET, &g_handle, data, length, 0);
    }
}
else if (status == kStatus_ENET_RxFrameError)
{
    Comment: Update the received buffer when a error frame is received.
    ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0);
}
```

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- data – The data buffer provided by user to store the frame which memory size should be at least “length”.
- length – The size of the data buffer which is still the length of the received frame.
- channel – The Rx DMA channel. Shall not be larger than 2.
- timestamp – The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

```
status_t ENET_GetRxFrame(ENET_Type *base, enet_handle_t *handle, enet_rx_frame_struct_t
*rxFrame, uint8_t channel)
```

Receives one frame in specified BD ring with zero copy.

This function will use the user-defined allocate and free callback. Every time application gets one frame through this function, driver will allocate new buffers for the BDs whose buffers have been taken by application.

Note: This function will drop current frame and update related BDs as available for DMA if new buffers allocating fails. Application must provide a memory pool including at least BD number + 1 buffers(+2 if enable double buffer) to make this function work normally.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- rxFrame – The received frame information structure provided by user.
- channel – The Rx DMA channel. Shall not be larger than 2.

Return values

- kStatus_Success – Succeed to get one frame and allocate new memory for Rx buffer.
- kStatus_ENET_RxFrameEmpty – There's no Rx frame in the BD.
- kStatus_ENET_RxFrameError – There's issue in this receiving. In this function, issue frame will be dropped.
- kStatus_ENET_RxFrameDrop – There's no new buffer memory for BD, dropped this frame.

status_t ENET_SendFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_tx_frame_struct_t* *txFrame, uint8_t channel)

Transmits an ENET frame.

Note: The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses input buffer for Tx, application should reclaim the buffer after Tx is over.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- txFrame – The Tx frame structure.
- channel – Channel to send the frame, same with queue index.

Return values

- kStatus_Success – Send frame succeed.
- kStatus_ENET_TxFrameBusy – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus_ENET_TxFrameBusy. Also need to pay attention to reclaim Tx frame after Tx is over.
- kStatus_ENET_TxFrameOverLen – Transmit frame length exceeds the 0x3FFF limit defined by the driver.

void ENET_ReclaimTxDescriptor(ENET_Type *base, *enet_handle_t* *handle, uint8_t channel)

Reclaim Tx descriptors. This function is used to update the Tx descriptor status and store the Tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- channel – The Tx DMA channel.

`void ENET_IRQHandler(ENET_Type *base, enet_handle_t *handle)`

The ENET IRQ handler.

Parameters

- `base` – ENET peripheral base address.
- `handle` – The ENET handler pointer.

`FSL_ENET_DRIVER_VERSION`

Defines the driver version.

`ENET_RXDESCRIP_RD_BUFF1VALID_MASK`

Defines for read format.

Buffer1 address valid.

`ENET_RXDESCRIP_RD_BUFF2VALID_MASK`

Buffer2 address valid.

`ENET_RXDESCRIP_RD_IOC_MASK`

Interrupt enable on complete.

`ENET_RXDESCRIP_RD_OWN_MASK`

Own bit.

`ENET_RXDESCRIP_WR_ERR_MASK`

Defines for write back format.

`ENET_RXDESCRIP_WR_PYLOAD_MASK`

`ENET_RXDESCRIP_WR_PTPMSGTYPE_MASK`

`ENET_RXDESCRIP_WR_PTPTYPE_MASK`

`ENET_RXDESCRIP_WR_PTPVERSION_MASK`

`ENET_RXDESCRIP_WR_PTPTSA_MASK`

`ENET_RXDESCRIP_WR_PACKETLEN_MASK`

`ENET_RXDESCRIP_WR_ERRSUM_MASK`

`ENET_RXDESCRIP_WR_TYPE_MASK`

`ENET_RXDESCRIP_WR_DE_MASK`

`ENET_RXDESCRIP_WR_RE_MASK`

`ENET_RXDESCRIP_WR_OE_MASK`

`ENET_RXDESCRIP_WR_RS0V_MASK`

`ENET_RXDESCRIP_WR_RS1V_MASK`

`ENET_RXDESCRIP_WR_RS2V_MASK`

`ENET_RXDESCRIP_WR_LD_MASK`

`ENET_RXDESCRIP_WR_FD_MASK`

`ENET_RXDESCRIP_WR_CTXT_MASK`

`ENET_RXDESCRIP_WR_OWN_MASK`

ENET_TXDESCRIP_RD_BL1_MASK

Defines for read format.

ENET_TXDESCRIP_RD_BL2_MASK

ENET_TXDESCRIP_RD_BL1(n)

ENET_TXDESCRIP_RD_BL2(n)

ENET_TXDESCRIP_RD_TTSE_MASK

ENET_TXDESCRIP_RD_IOC_MASK

ENET_TXDESCRIP_RD_FL_MASK

ENET_TXDESCRIP_RD_FL(n)

ENET_TXDESCRIP_RD_CIC(n)

ENET_TXDESCRIP_RD_TSE_MASK

ENET_TXDESCRIP_RD_SLOT(n)

ENET_TXDESCRIP_RD_SAIC(n)

ENET_TXDESCRIP_RD_CPC(n)

ENET_TXDESCRIP_RD_LDFD(n)

ENET_TXDESCRIP_RD_LD_MASK

ENET_TXDESCRIP_RD_FD_MASK

ENET_TXDESCRIP_RD_CTXT_MASK

ENET_TXDESCRIP_RD_OWN_MASK

ENET_TXDESCRIP_WB_TTSS_MASK

Defines for write back format.

ENET_ABNORM_INT_MASK

ENET_NORM_INT_MASK

__IO uint32_t rdes0

Receive descriptor 0

__IO uint32_t rdes1

Receive descriptor 1

__IO uint32_t rdes2

Receive descriptor 2

__IO uint32_t rdes3

Receive descriptor 3

__IO uint32_t tdes0

Transmit descriptor 0

__IO uint32_t tdes1

Transmit descriptor 1

__IO uint32_t tdes2

Transmit descriptor 2

__IO uint32_t tdes3

Transmit descriptor 3

void *buffer1

The first buffer address in the descriptor.

uint32_t bytes1

The bytes in the fist buffer.

void *buffer2

The second buffer address in the descriptor.

uint32_t bytes2

The bytes in the second buffer.

uint32_t framelen

The length of the frame to be transmitted.

bool intEnable

Interrupt enable flag.

bool tsEnable

The timestamp enable.

enet_tx_offload_t txOffloadOps

The Tx checksum offload option, only vaild for Queue 0.

enet_desc_flag_t flag

The flag of this tx desciriptor, see “enet_qos_desc_flag”.

uint8_t slotNum

The slot number used for AV mode only.

uint64_t second

Second.

uint32_t nanosecond

Nanosecond.

void *context

User specified data, could be buffer address for free

bool isTsAvail

Flag indicates timestamp available status

enet_ptp_time_t timeStamp

Timestamp of frame

enet_tx_reclaim_info_t *txDirtyBase

Dirty buffer descriptor base address pointer.

uint16_t txGenIdx

Tx generate index.

uint16_t txConsumIdx

Tx consume index.

uint16_t txRingLen

Tx ring length.

bool isFull

Tx ring is full flag, add this parameter to avoid waste one element.

`uint8_t rxRingLen`
The length of receive buffer descriptor ring.

`uint8_t txRingLen`
The length of transmit buffer descriptor ring.

`enet_tx_bd_struct_t *txDescStartAddrAlign`
Aligned transmit descriptor start address.

`enet_tx_bd_struct_t *txDescTailAddrAlign`
Aligned transmit descriptor tail address.

`enet_tx_reclaim_info_t *txDirtyStartAddr`
Start address of the dirty Tx frame information.

`enet_rx_bd_struct_t *rxDescStartAddrAlign`
Aligned receive descriptor start address.

`enet_rx_bd_struct_t *rxDescTailAddrAlign`
Aligned receive descriptor tail address.

`uint32_t *rxBufferStartAddr`
Start address of the Rx buffers.

`uint32_t rxBuffSizeAlign`
Aligned receive data buffer size.

`enet_dma_tx_sche_t dmaTxSche`
Transmit arbitration.

`enet_dma_burstlen_t burstLen`
Burst len for the queue 1.

`uint8_t txdmaChnWeight[ENET_MTL_QUEUE_COUNT]`
Transmit channel weight.

`enet_mtl_multiqueue_txsche_t mtltxSche`
Transmit schedule for multi-queue.

`enet_mtl_multiqueue_rxsche_t mtlrxSche`
Receive schedule for multi-queue.

`uint8_t rxqueweight[ENET_MTL_QUEUE_COUNT]`
Refer to the MTL RxQ Control register.

`uint32_t txqueweight[ENET_MTL_QUEUE_COUNT]`
Refer to the MTL TxQ Quantum Weight register.

`uint8_t rxqueuePrio[ENET_MTL_QUEUE_COUNT]`
Receive queue priority.

`uint8_t txqueuePrio[ENET_MTL_QUEUE_COUNT]`
Refer to Transmit Queue Priority Mapping register.

`enet_mtl_rxqueuemap_t mtlrxQuemap`
Rx queue DMA Channel mapping.

`uint16_t specialControl`
The logical or of `enet_special_config_t`

`uint32_t ptpClkHz`
The PTP module source clock.

enet_multiqueue_config_t *multiqueueCfg

Use both Tx/Rx queue(dma channel) 0 and 1.

uint32_t interrupt

MAC interrupt source. A logical OR of enet_dma_interrupt_enable_t and enet_mac_interrupt_enable_t.

enet_mii_mode_t miiMode

MII mode.

enet_mii_speed_t miiSpeed

MII Speed.

enet_mii_duplex_t miiDuplex

MII duplex.

uint16_t pauseDuration

Used in the Tx flow control frame, only valid when kENET_FlowControlEnable is set.

enet_rx_alloc_callback_t rxBuffAlloc

Callback to alloc memory, must be provided for zero-copy Rx.

enet_rx_free_callback_t rxBuffFree

Callback to free memory, must be provided for zero-copy Rx.

enet_tx_bd_struct_t *txBdBase

Buffer descriptor base address pointer.

uint16_t txGenIdx

Tx generate index.

uint16_t txConsumIdx

Tx consum index.

volatile uint16_t txDescUsed

Tx descriptor used number.

uint16_t txRingLen

Tx ring length.

enet_rx_bd_struct_t *rxBdBase

Buffer descriptor base address pointer.

uint16_t rxGenIdx

The current available receive buffer descriptor pointer.

uint16_t rxRingLen

Receive ring length.

uint32_t rxBuffSizeAlign

Receive buffer size.

bool multiQueEnable

Multi-queue enable status.

bool doubleBuffEnable

The double buffer enable status.

bool rxintEnable

Rx interrupt enable status.

enet_rx_bd_ring_t rxBdRing[ENET_MTL_QUEUE_COUNT]

Receive buffer descriptor.

enet_tx_bd_ring_t txBdRing[ENET_MTL_QUEUE_COUNT]

Transmit buffer descriptor.

enet_tx_dirty_ring_t txDirtyRing[ENET_MTL_QUEUE_COUNT]

Transmit dirty buffers addresses.

uint32_t *rxBufferStartAddr[ENET_MTL_QUEUE_COUNT]

The Init-Rx buffers used for reinit corrupted BD due to write-back operation.

uint32_t txLenLimitation[ENET_MTL_QUEUE_COUNT]

Tx frame length limitation.

enet_callback_t callback

Callback function.

void *userData

Callback function parameter.

enet_rx_alloc_callback_t rxBuffAlloc

Callback to alloc memory, must be provided for zero-copy Rx.

enet_rx_free_callback_t rxBuffFree

Callback to free memory, must be provided for zero-copy Rx.

void *buffer

The buffer stores the whole or partial frame.

uint16_t length

The byte length of this buffer.

bool isTsAvail

Rx frame timestamp is available or not.

enet_ptp_time_t timestamp

The nanosecond part timestamp of this Rx frame.

bool statsDribbleErr

The received packet has a non-integer multiple of bytes (odd nibbles).

bool statsRxErr

Receive error.

bool statsOverflowErr

Rx FIFO overflow error.

bool statsWatchdogTimeoutErr

Receive watchdog timeout.

bool statsGaintPacketErr

Receive error.

bool statsRxFcsErr

Receive CRC error.

enet_buffer_struct_t *rxBuffArray

Rx frame buffer structure.

uint16_t totLen

Rx frame total length.

enet_rx_frame_attribute_t rxAttribute

Rx frame attribute structure.

enet_rx_frame_error_t rxFrameError

Rx frame error.

uint8_t intEnable

Enable interrupt every time one BD is completed.

uint8_t tsEnable

Transmit timestamp enable.

uint8_t slotNum

Slot number control bits in AV mode.

enet_tx_offload_t txOffloadOps

Tx checksum offload option.

enet_buffer_struct_t *txBuffArray

Tx frame buffer structure.

uint32_t txBuffNum

Buffer number of this Tx frame.

enet_tx_config_struct_t txConfig

Tx extra configuration.

void *context

Driver reclaims and gives it in Tx over callback.

enet_vlan_tpid_t tpid

VLAN TPID.

uint16_t pcp

VLAN Priority.

uint16_t dei

Drop Eligible indicator.

uint16_t vid

VLAN Identifier.

bool txDescVlan

Use VLAN configuration in Tx descriptor.

enet_vlan_tag_t tag

VLAN Tag.

enet_vlan_ops_t ops

VLAN operations.

bool svlanEnable

The MAC transmitter and receiver consider the S-VLAN packets.

bool vlanInverseMatch

True: Marks frames without matching as match, False: Marks matched frames.

bool vidComparison

Only takes VLAN VID as match.

bool disableVlanTypeCheck

Not check C-VLAN and S-VLAN.

bool doubleVlanEnable

Enable the inner VLAN operations.

bool innerVlanFilterMatch

Takes Inner VLAN as match.

bool outerTagInRxStatus

Set outer VLAN in Rx Status.

bool innerTagInRxStatus

Set inner VLAN in Rx Status.

enet_vlan_tag_t rxVlanTag

VLAN tag for Rx match.

enet_vlan_strip_t rxOuterVlanStrip

Outer VLAN Rx strip operation.

enet_vlan_strip_t rxInnerVlanStrip

Inner VLAN Rx strip operation.

struct _enet_rx_bd_struct

#include <fsl_enet.h> Defines the receive descriptor structure. It has the read-format and write-back format structures. They both have the same size with different region definition. So we define common name as the receive descriptor structure. When initialize the buffer descriptors, read-format region mask bits should be used. When Rx frame has been in the buffer descriptors, write-back format region store the Rx result information.

struct _enet_tx_bd_struct

#include <fsl_enet.h> Defines the transmit descriptor structure. It has the read-format and write-back format structure. They both have the same size with different region definition. So we define common name as the transmit descriptor structure. When initialize the buffer descriptors for Tx, read-format region mask bits should be used. When frame has been transmitted, write-back format region store the Tx result information.

struct _enet_tx_bd_config_struct

#include <fsl_enet.h> Defines the Tx BD configuration structure.

struct _enet_ptp_time

#include <fsl_enet.h> Defines the ENET PTP time stamp structure.

struct enet_tx_reclaim_info

#include <fsl_enet.h> Defines the Tx reclaim information structure.

struct _enet_tx_dirty_ring

#include <fsl_enet.h> Defines the ENET transmit dirty addresses ring/queue structure.

struct _enet_buffer_config

#include <fsl_enet.h> Defines the buffer descriptor configure structure.

Notes:

- a. The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.
- b. The recommended minimum Tx/Rx ring length is 4.
- c. The Tx/Rx descriptor tail address shall be the address pointer to the address just after the end of the last descriptor. Because only the descriptors between the start address and the tail address will be used by DMA.
- d. The descriptor address is the start address of all used contiguous memory. For example, the rxDescStartAddrAlign is the start address of rxRingLen contiguous descriptor memory for Rx descriptor ring 0.

- e. The “*rxBufferstartAddr” is the first element of rxRingLen (2*rxRingLen for double buffers) Rx buffers. It means the *rxBufferStartAddr is the Rx buffer for the first descriptor the *rxBufferStartAddr + 1 is the Rx buffer for the second descriptor or the Rx buffer for the second buffer in the first descriptor. So please make sure the rxBufferStartAddr is the address of a rxRingLen or 2*rxRingLen array.

struct enet_multiqueue_config

#include <fsl_enet.h> Defines the configuration when multi-queue is used.

struct __enet_config

#include <fsl_enet.h> Defines the basic configuration structure for the ENET device.

Note:

- a. Default the signal queue is used so the “multiqueueCfg” is set default with NULL. Set the pointer with a valid configuration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

struct __enet_tx_bd_ring

#include <fsl_enet.h> Defines the ENET transmit buffer descriptor ring/queue structure.

struct __enet_rx_bd_ring

#include <fsl_enet.h> Defines the ENET receive buffer descriptor ring/queue structure.

struct __enet_handle

#include <fsl_enet.h> Defines the ENET handler structure.

struct __enet_buffer_struct

#include <fsl_enet.h>

struct __enet_rx_frame_attribute_struct

#include <fsl_enet.h> Rx frame attribute structure.

struct __enet_rx_frame_error

#include <fsl_enet.h> Defines the Rx frame error structure.

struct __enet_rx_frame_struct

#include <fsl_enet.h> Defines the Rx frame data structure.

struct __enet_tx_config_struct

#include <fsl_enet.h>

struct __enet_tx_frame_struct

#include <fsl_enet.h>

struct __enet_vlan_tag

#include <fsl_enet.h> Ethernet VLAN Tag.

struct __enet_vlan_tx_config

#include <fsl_enet.h> Ethernet VLAN configuration for Tx.

struct __enet_vlan_ctrl

#include <fsl_enet.h> Ethernet VLAN control.

2.58 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:

- Disable all LVD's and HVD's in low power mode;
 - Disable Glitch detect in low power mode;
 - Configure LDO's and DCDC to low drive strength in low power mode;
 - 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_CORELDVVoltageSetFail` – 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_GetActiveModeCoreLDVVDDVoltageLevel(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_CORELDOVoltageSetFail` – Fail to change Core LDO voltage level.

```
status_t SPC_SetLowPowerModeCoreLDORegulatorVoltageLevel(SPC_Type *base,  
*spc_core_ldo_voltage_level_t  
*voltageLevel)
```

Set Core LDO VDD Regulator Voltage level in Low power mode.

Note: If CORE LDO's drive strength is set to Normal, the CORE LDO VDD regulator voltage in active mode and low power mode must be same.

Note: Voltage level for the CORE LDO in low power mode can only be changed when the CORE LDO Drive Strength set as Normal.

Parameters

- `base` – SPC peripheral base address.
- `voltageLevel` – Voltage level of CORE LDO Regulator in Low power mode, please refer to `spc_core_ldo_voltage_level_t`.

Return values

- `kStatus_SPC_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.

[illegible]

Set DCDC CORE Regulator voltage level in Active mode.

Note: When changing the DCDC output voltage level, take care to change the CORE LDO voltage level.

Parameters

- `base` – SPC peripheral base address.
- `voltageLevel` – Specify the DCDC_CORE Regulator voltage level, please refer to `spc_dcdc_voltage_level_t`.

```
static inline spc_dcdc_voltage_level_t SPC_GetActiveModeDCDCRegulatorVoltageLevel(SPC_Type  
*base)
```

Get DCDC_CORE Regulator voltage level in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator voltage level, please refer to `spc_dcdc_voltage_level_t`.

```
status_t SPC_SetActiveModeDCDCRegulatorDriveStrength(SPC_Type *base,
                                                    spc_dcdc_drive_strength_t
                                                    driveStrength)
```

Set DCDC_CORE Regulator drive strength in Active mode.

Note: To set DCDC drive strength as Normal, the bandgap must be enabled.

Parameters

- base – SPC peripheral base address.
- driveStrength – Specify the DCDC_CORE regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

Return values

- `kStatus_Success` – Set DCDC_CORE Regulator drive strength in Active mode successfully.
- `kStatus_SPC_BandgapModeWrong` – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

```
static inline spc_dcdc_drive_strength_t SPC_GetActiveModeDCDCRegulatorDriveStrength(SPC_Type
                                                                                      *base)
```

Get DCDC_CORE Regulator drive strength in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

```
status_t SPC_SetLowPowerModeDCDCRegulatorConfig(SPC_Type *base, const
                                                  spc_lowpower_mode_dcdc_option_t
                                                  *option)
```

Configs DCDC_CORE Regulator in Low power modes.

Note: If DCDC_CORE Drive Strength is set to Normal, the Bandgap mode in Low Power mode must be programmed to a value that enables the Bandgap.

Note: In Deep Power Down mode, DCDC regulator is always turned off.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the `spc_lowpower_mode_dcdc_option_t` structure.

Return values

- `kStatus_Success` – Config DCDC regulator in low power mode successfully.
- `kStatus_SPC_Busy` – The SPC instance is busy to execute any type of power mode transition.

- `kStatus_SPC_BandgapModeWrong` – The bandgap mode setting in Low Power mode is wrong.

```
status_t SPC_SetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type *base,  
                                                         spc_dcdc_drive_strength_t  
                                                         driveStrength)
```

Set DCDC_CORE Regulator drive strength in Low power mode.

Note: To set drive strength as normal, the bandgap must be enabled.

Parameters

- `base` – SPC peripheral base address.
- `driveStrength` – Specify the DCDC_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

Return values

- `kStatus_Success` – Set DCDC_CORE Regulator drive strength in Low power mode successfully.
- `kStatus_SPC_BandgapModeWrong` – Set DCDC_CORE Regulator drive strength to Normal, the Bandgap must be enabled.

```
static inline spc_dcdc_drive_strength_t SPC_GetLowPowerModeDCDCRegulatorDriveStrength(SPC_Type  
                                                                                       *base)
```

Get DCDC_CORE Regulator drive strength in Low power mode.

Parameters

- `base` – SPC peripheral base address.

Returns

DCDC_CORE Regulator drive strength, please refer to `spc_dcdc_drive_strength_t`.

```
static inline void SPC_SetLowPowerModeDCDCRegulatorVoltageLevel(SPC_Type *base,  
                                                                spc_dcdc_voltage_level_t  
                                                                voltageLevel)
```

Set DCDC_CORE Regulator voltage level in Low power mode.

- a. Configure `ACTIVE_CFG[DCDC_VDD_LVL]` to same level programmed in #1.

Note: To change DCDC level in Low-Power mode:

- a. Configure `LP_CFG[DCDC_VDD_LVL]` to desired level;
 - b. Configure `LP_CFG[DCDC_VDD_DS]` to low driver strength;
-

Note: After invoking this function, the voltage level in active mode(wakeup from low power modes) also changed, if it is necessary, please invoke `SPC_SetActiveModeDCDCRegulatorVoltageLevel()` to change to desired voltage level.

Parameters

- `base` – SPC peripheral base address.

- voltageLevel – Specify the DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

static inline spc_dcdc_voltage_level_t SPC_GetLowPowerModeDCDCRegulatorVoltageLevel(SPC_Type *base)

Get DCDC_CORE Regulator voltage level in Low power mode.

Parameters

- base – SPC peripheral base address.

Returns

DCDC_CORE Regulator voltage level, please refer to spc_dcdc_voltage_level_t.

FSL_SPC_DRIVER_VERSION

SPC driver version 2.12.0.

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_CORELDOVoltageWrong

Core LDO voltage is wrong.

enumerator kStatus_SPC_CORELDOVoltageSetFail

Core LDO voltage set fail.

enumerator kStatus_SPC_CORELDOVoltageDetectWrong

Settings of CORE_LDO voltage detection is not allowed.

enumerator kStatus_SPC_DCDCCoreLdoVoltageMisMatch

Target voltage level of DCDC not equal to CORE_LDO.

enum _spc_voltage_detect_flags

Voltage Detect Status Flags.

Values:

enumerator kSPC_IOVDDHighVoltageDetectFlag
IO VDD High-Voltage detect flag.

enumerator kSPC_IOVDDLowVoltageDetectFlag
IO VDD Low-Voltage detect flag.

enumerator kSPC_SystemVDDHighVoltageDetectFlag
System VDD High-Voltage detect flag.

enumerator kSPC_SystemVDDLowVoltageDetectFlag
System VDD Low-Voltage detect flag.

enumerator kSPC_CoreVDDHighVoltageDetectFlag
Core VDD High-Voltage detect flag.

enumerator kSPC_CoreVDDLowVoltageDetectFlag
Core VDD Low-Voltage detect flag.

enum _spc_power_domains
SPC power domain isolation status.

Note: Some devices(such as MCXA family) do not contain WAKE Power Domain, please refer to the reference manual to check.

Values:

enumerator kSPC_MAINPowerDomainRetain
Peripherals and IO pads retain in MAIN Power Domain.

enumerator kSPC_WAKEPowerDomainRetain
Peripherals and IO pads retain in WAKE Power Domain.

enum _spc_analog_module_control
The enumeration of all analog module that can be controlled by SPC in active or low-power modes.

Note: Enumerations may not suitable for all devices, please check the specific device's RM for supported analog modules.

Values:

enumerator kSPC_controlVref
Enable/disable VREF in active or low-power modes.

enumerator kSPC_controlUsb3vDet
Enable/disable USB3V_Det in active or low-power modes.

enumerator kSPC_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_CoreLDO_RetentionVoltage as instead.

enumerator kSPC_CoreLDO_RetentionVoltage

Core LDO VDD regulator regulate to retention voltage, please note that only useful in low power modes and not all devices support this options please refer to devices' RM for details.

enumerator kSPC_CoreLDO_MidDriveVoltage

Core LDO VDD regulator regulate to Mid Drive Voltage.

enumerator kSPC_CoreLDO_NormalVoltage

Core LDO VDD regulator regulate to Normal Voltage.

enumerator kSPC_CoreLDO_OverDriveVoltage

Core LDO VDD regulator regulate to overdrive Voltage.

enum _spc_core_ldo_drive_strength

CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

Values:

enumerator kSPC_CoreLDO_LowDriveStrength

Core LDO VDD regulator Drive Strength set to low.

enumerator kSPC_CoreLDO_NormalDriveStrength

Core LDO VDD regulator Drive Strength set to Normal.

enum _spc_low_voltage_level_select

IO VDD Low-Voltage Level Select.

Values:

enumerator kSPC_LowVoltageNormalLevel

Deprecated:

, please use kSPC_LowVoltageHighRange as instead.

enumerator kSPC_LowVoltageSafeLevel

Deprecated:

, please use kSPC_LowVoltageLowRange as instead.

enumerator kSPC_LowVoltageHighRange

High range LVD threshold.

enumerator kSPC_LowVoltageLowRange

Low range LVD threshold.

enum _spc_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 HVDRResetEnable

CORE/SYS/IO VDD High Voltage Detect reset enable.

bool LVDInterruptEnable

CORE/SYS/IO VDD Low Voltage Detect interrupt enable.

bool LVDRResetEnable

CORE/SYS/IO VDD Low Voltage Detect reset enable.

struct *_spc_core_voltage_detect_config*

#include <fsl_spc.h> Core Voltage Detect configuration.

Public Members

spc_voltage_detect_option_t option

Core VDD Voltage Detect option.

struct *_spc_system_voltage_detect_config*

#include <fsl_spc.h> System Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option
System VDD Voltage Detect option.

spc_low_voltage_level_select_t level

Deprecated:

, reserved for all devices, will removed in next release.

struct *_spc_io_voltage_detect_config*
#include <fsl_spc.h> IO Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option
IO VDD Voltage Detect option.

spc_low_voltage_level_select_t level
IO VDD Low-voltage level selection.

struct *_spc_active_mode_regulators_config*
#include <fsl_spc.h> Active mode configuration.

Public Members

spc_bandgap_mode_t bandgapMode
Specify bandgap mode in active mode.

bool *lpBuff*
Enable/disable CMP bandgap buffer.

spc_active_mode_dcdc_option_t DCDCOption
Specify DCDC configurations in active mode.

spc_active_mode_sys_ldo_option_t SysLDOOption
Specify System LDO configurations in active mode.

spc_active_mode_core_ldo_option_t CoreLDOOption
Specify Core LDO configurations in active mode.

struct *_spc_lowpower_mode_regulators_config*
#include <fsl_spc.h> Low Power Mode configuration.

Public Members

bool *lpIREF*
Enable/disable low power IREF in low power modes.

spc_bandgap_mode_t bandgapMode
Specify bandgap mode in low power modes.

bool *lpBuff*
Enable/disable CMP bandgap buffer in low power modes.

bool *CoreIVS*
Enable/disable CORE VDD internal voltage scaling.

spc_lowpower_mode_dcdc_option_t DCDCOption

Specify DCDC configurations in low power modes.

spc_lowpower_mode_sys_ldo_option_t SysLDOOption

Specify system LDO configurations in low power modes.

spc_lowpower_mode_core_ldo_option_t CoreLDOOption

Specify core LDO configurations in low power modes.

2.59 NBOOT Driver

`typedef uint32_t nboot_sb4_fwtype_t`

`typedef uint32_t nboot_root_key_usage_t`

`typedef uint32_t nboot_root_key_revocation_t`

`typedef uint32_t nboot_root_key_type_and_length_t`

`typedef uint32_t nboot_soc_lifecycle_t`

`typedef uint32_t nboot_status_t`

Type for nboot status codes.

`typedef uint64_t nboot_status_protected_t`

Type for nboot protected status codes.

`FSL_ROMAPI_NBOOT_DRIVER_VERSION`

ROMAPI_NBOOT driver version 2.0.0.

`NBOOT_SB4_MANIFEST_MAX_SIZE_IN_BYTES`

`NBOOT_SB4_BLOCK_MAX_SIZE_IN_BYTES`

`NBOOT_SB4_GET_INITIAL_FETCH_SIZE`

`NBOOT_SB4_FWTYPE_DEFAULT`

`NBOOT_SB4_FWTYPE_NXP_MFW`

`NBOOT_SB4_FWTYPE_OEM_MFW`

`kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA_ImageKey_FwKey`

NBOOT type for the root key usage.

This type defines the NBOOT root key usage

`kNBOOT_RootKeyUsage_DebugCA`

`kNBOOT_RootKeyUsage_ImageCA_FwCA`

`kNBOOT_RootKeyUsage_DebugCA_ImageCA_FwCA`

`kNBOOT_RootKeyUsage_ImageKey_FwKey`

`kNBOOT_RootKeyUsage_ImageKey`

`kNBOOT_RootKeyUsage_FwKey`

`kNBOOT_RootKeyUsage_Unused`

`NBOOT_ROOT_CERT_COUNT`

kNBOOT_RootKey_Enabled

NBOOT type for the root key revocation.

This type defines the NBOOT root key revocation

kNBOOT_RootKey_Revoked

kNBOOT_RootKey_Ecdsa_P256

NBOOT type specifying the elliptic curve to be used.

This type defines the elliptic curve type and length

kNBOOT_RootKey_Ecdsa_P384

kNBOOT_RootKey_MiDsa_87

kNBOOT_RootKey_Ecdsa_P384_MiDsa_87

nboot_lc_nxpBlank

Enumeration for SoC Lifecycle.

nboot_lc_nxpFab

nboot_lc_nxpDev

nboot_lc_nxpProvisioned

nboot_lc_oemOpen

nboot_lc_oemSecureWorld

nboot_lc_oemClosed

nboot_lc_oemLocked

nboot_lc_oemFieldReturn

nboot_lc_nxpFieldReturn

nboot_lc_shredded

MCUX_CSSL_FP_FUNCID_NBOOT_ContextInit

Enumeration for MCUX_CSSL_FP_FUNCID.

MCUX_CSSL_FP_FUNCID_NBOOT_ContextDeinit

MCUX_CSSL_FP_FUNCID_NBOOT_ContextSetUuid

MCUX_CSSL_FP_FUNCID_NBOOT_Sb4LoadManifest

MCUX_CSSL_FP_FUNCID_NBOOT_Sb4LoadBlock

MCUX_CSSL_FP_FUNCID_NBOOT_SB4CheckAuthenticityAndCompleteness

MCUX_CSSL_FP_FUNCID_NBOOT_ImgAuthenticate

NBOOT_CONTEXT_RTF_OFFSET

kStatus_NBOOT_Success

Operation completed successfully.

kStatus_NBOOT_Fail

Operation failed.

kStatus_NBOOT_InvalidArgument

Invalid argument passed to the function.

kStatus_NBOOT_RequestTimeout

Operation timed out.

kStatus_NBOOT_KeyNotLoaded

The requested key is not loaded.

kStatus_NBOOT_AuthFail

Authentication failed.

kStatus_NBOOT_OperationNotAvailable

Operation not available on this HW.

kStatus_NBOOT_KeyNotAvailable

Key is not available.

kStatus_NBOOT_IvCounterOverflow

Overflow of IV counter (PRINCE/IPED).

kStatus_NBOOT_SelftestFail

FIPS self-test failure.

kStatus_NBOOT_InvalidDataFormat

Invalid data format for example antipole

kStatus_NBOOT_IskCertUserDataTooBig

Size of User data in ISK certificate is greater than 96 bytes

kStatus_NBOOT_IskCertSignatureOffsetTooSmall

Signature offset in ISK certificate is smaller than expected

kStatus_NBOOT_MemcpyFail

Unexpected error detected during nboot_memcpy()

kStatus_NBOOT_RegionIsLocked

IPED/NPX region is locked

kStatus_NBOOT_SB3_Hashing

nboot_sb3_load_block () background hashing started

kStatus_NBOOT_SB3_Decrypting

nboot_sb3_load_block () background decrypting started

kStatus_NBOOT_SB4_Hashing

nboot_sb4_load_block () background hashing started

kStatus_NBOOT_SB4_Decrypting

nboot_sb4_load_block () background decrypting started

enum_nboot_hash_algo_t

Algorithm used for nboot HASH operation.

Values:

enumerator kHASH_Sha1

SHA_1

enumerator kHASH_Sha256

SHA_256

enumerator kHASH_Sha512

SHA_512

```
enumerator kHASH_Aes
    AES
enumerator kHASH_AesIcb
    AES_ICB
typedef enum _nboot_hash_algo_t nboot_hash_algo_t
    Algorithm used for nboot HASH operation.
uint32_t sc
uint32_t scAp
uint32_t configData[13]
uint8_t opaque[0x00000858u]
nboot_root_key_revocation_t soc_rootKeyRevocation[(4u)]
    Provided by caller based on NVM information in CFPA: ROTKH_REVOKE
uint32_t soc_imageKeyRevocation
    Provided by caller based on NVM information in CFPA: IMAGE_KEY_REVOKE
uint32_t soc_rkh[(12u)]
    Provided by caller based on NVM information in CMPA: ROTKH (hash of hashes)
    In case of kNBOOT_RootKey_Ecdsa_P384, sock_rkh[0..11] are used In case of kN-
    BOOT_RootKey_Ecdsa_P256, sock_rkh[0..7] are used In case of kNBOOT_RootKey_MlDsa_87,
    sock_rkh[0..11] are used
uint32_t soc_rkh_1[(12u)]
    Provided by caller based on NVM information in CMPA: ROTKH (hash of hashes) In case of
    kNBOOT_RootKey_MlDsa_87, sock_rkh[0..11] are used
uint32_t soc_numberOfRootKeys
nboot_root_key_usage_t soc_rootKeyUsage[(4u)]
nboot_root_key_type_and_length_t soc_rootKeyTypeAndLength
nboot_soc_lifecycle_t soc_lifecycle
nboot_rot_auth_parms_t soc_RoTNVM
uint32_t soc_trustedFirmwareVersion
    Provided by caller based on NVM information in CFPA: Secure_FW_Version
nboot_rot_auth_parms_t soc_RoTNVM
uint32_t soc_trustedFirmwareVersion
    trusted information originated from CFPA and NMPA Provided by caller based on NVM
    information in CFPA: Secure_FW_Version
uint32_t maxBlockSize
    Provided by caller: supported maximal data block size (complete block)
uint8_t pckBlob[48]
nboot_mem_crypt_engine_type_t configId
uint32_t startAddress
uint32_t endAddress
uint64_t subregion
```

uint32_t ivEraseCounter

uint8_t regionLock

nboot_mem_crypt_engine_type_t configId

uint32_t startAddress

uint32_t endAddress

uint32_t ivEraseCounter

uint8_t regionFlags

uint32_t aad

nboot_npx_region_config_t npxConfig

nboot_iped_region_config_t ipedConfig

NBOOT_UUID_SIZE_IN_WORD

The size of the UUID.

NBOOT_UUID_SIZE_IN_BYTE

NBOOT_PUF_AC_SIZE_IN_BYTE

The size of the PUF activation code.

NBOOT_PUF_KC_SIZE_IN_BYTE

The size of the PUF key code.

NBOOT_KEY_STORE_SIZE_IN_BYTE

The size of the key store.

NBOOT_ROOT_ROT_KH_SIZE_IN_WORD

The size of the root of trust key table hash.

NBOOT_ROOT_ROT_KH_SIZE_IN_BYTE

NBOOT_KEY_BLOB_SIZE_IN_BYTE_256

The size of the blob with Part Common Key.

NBOOT_KEY_BLOB_SIZE_IN_BYTE_384

NBOOT_KEY_BLOB_SIZE_IN_BYTE_MAX

NBOOT_DBG_AUTH_DBG_STATE_MASK

The mask of the value of the debug state .

NBOOT_DBG_AUTH_DBG_STATE_SHIFT

The shift inverted value of the debug state.

NBOOT_DBG_AUTH_DBG_STATE_ALL_DISABLED

The value with all debug feature disabled.

NBOOT_DICE_CSR_SIZE_IN_WORD

NBOOT_DICE_CSR_SIZE_IN_BYTES

NBOOT_DICE_CSR_ADDRESS

NBOOT_IPED_IV_OFFSET

The offset for the PRCINE/IPED erase region return by nboot mem checker.

NBOOT_IMAGE_CMAC_UPDATE_NONE
NBOOT_IMAGE_CMAC_UPDATE_INDEX0
NBOOT_IMAGE_CMAC_UPDATE_INDEX1
NBOOT_IMAGE_CMAC_UPDATE_BOTH
NBOOT_IMAGE_CMAC_UPDATE_MASK
NBOOT_CMPA_CMAC_UPDATE_MASK
NBOOT_CMPA_CMAC_UPDATE_SHIFT
NBOOT_CMPA_UPDATE_CMAC_PFR
NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_SECURE
NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_CLOSE
NBOOT_CMPA_UPDATE_CMAC_PFR_OTP_OEM_LOCKED
EFUSE_PART_CFG_FUSE_INDEX

2.60 This type defines status return values used by NBOOT functions that are not easily

kStatus_NBOOT_Success
Operation completed successfully.

kStatus_NBOOT_Fail
Operation failed.

kStatus_NBOOT_InvalidArgument
Invalid argument passed to the function.

kStatus_NBOOT_RequestTimeout
Operation timed out.

kStatus_NBOOT_KeyNotLoaded
The requested key is not loaded.

kStatus_NBOOT_AuthFail
Authentication failed.

kStatus_NBOOT_OperationNotAvailable
Operation not available on this HW.

kStatus_NBOOT_KeyNotAvailable
Key is not available.

kStatus_NBOOT_IvCounterOverflow
Overflow of IV counter (PRINCE/IPED).

kStatus_NBOOT_SelftestFail
FIPS self-test failure.

kStatus_NBOOT_InvalidDataFormat
Invalid data format for example antipole

kStatus_NBOOT_IskCertUserDataTooBig

Size of User data in ISK certificate is greater than 96 bytes

kStatus_NBOOT_IskCertSignatureOffsetTooSmall

Signature offset in ISK certificate is smaller than expected

kStatus_NBOOT_MemcpyFail

Unexpected error detected during nboot_memcpy()

kStatus_NBOOT_RegionIsLocked

IPED/NPX region is locked

kStatus_NBOOT_SB3_Hashing

nboot_sb3_load_block () background hashing started

kStatus_NBOOT_SB3_Decrypting

nboot_sb3_load_block () background decrypting started

kStatus_NBOOT_SB4_Hashing

nboot_sb4_load_block () background hashing started

kStatus_NBOOT_SB4_Decrypting

nboot_sb4_load_block () background decrypting started

enum __nboot_bool

Boolean type for the NBOOT functions.

This type defines boolean values used by NBOOT functions that are not easily disturbed by Fault Attacks

Values:

enumerator kNBOOT_TRUE

Value for TRUE.

enumerator kNBOOT_TRUE256

Value for TRUE when P256 was used to sign the image.

enumerator kNBOOT_TRUE384

Value for TRUE when P384 was used to sign the image.

enumerator kNBOOT_FALSE

Value for FALSE.

enumerator kNBOOT_OperationAllowed

enumerator kNBOOT_OperationDisallowed

enum nboot_mem_crypt_iped_mode_select_t

Values:

enumerator kNBOOT_MemCrypt12Rounds

enumerator kNBOOT_MemCrypt22Rounds

enumerator kNBOOT_MemCryptFullyPipelined

enumerator kNBOOT_MemCryptNotFullyPipelined

enum nboot_mem_crypt_engine_type_t

Values:

enumerator kNBOOT_MemCryptNpx

enumerator `kNBOOT_MemCryptIped`

typedef struct *nboot_secure_counter* nboot_secure_counter_t

Data structure holding secure counter value used by nboot library.

typedef struct *nboot_trng_cfg_load* nboot_trng_cfg_load_t

typedef struct *nboot_context_struct* nboot_context_t

typedef struct *nboot_rot_auth_parms* nboot_rot_auth_parms_t

NBOOT type for the root of trust parameters.

This type defines the NBOOT root of trust parameters

typedef struct *nboot_img_auth_parms_struct* nboot_img_auth_parms_t

Data structure holding input arguments to POR secure boot (authentication) algorithm. Shall be read from SoC trusted NVM or SoC fuses.

typedef struct *nboot_sb4_load_manifest_parms_struct* nboot_sb4_load_manifest_parms_t

manifest loading parameters

This type defines the NBOOT SB4 manifest loading parameters

typedef enum *nboot_bool* nboot_bool_t

Boolean type for the NBOOT functions.

This type defines boolean values used by NBOOT functions that are not easily disturbed by Fault Attacks

typedef uint32_t nboot_mem_crypt_region_t

typedef uint32_t nboot_mem_crypt_operation_t

typedef struct *nboot_npx_region_config_struct* nboot_npx_region_config_t

typedef struct *nboot_iped_region_config_struct* nboot_iped_region_config_t

typedef union *nboot_mem_crypt_region_config_union* nboot_mem_crypt_region_config_t

nboot_status_protected_t NBOOT_ContextInit(*nboot_context_t* *context)

The function is used for initializing of the nboot context data structure. It should be called prior to any other calls of nboot API.

Parameters

- `nbootCtx` – Pointer to `nboot_context_t` structure.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

nboot_status_protected_t NBOOT_ContextDeinit(*nboot_context_t* *context)

The function is used to deinitialize nboot context data structure. Its contents are overwritten with random data so that any sensitive data does not remain in memory.

Parameters

- `context` – Pointer to `nboot_context_t` structure.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

```
nboot_status_protected_t NBOOT_ContextSetUuid(nboot_context_t *context, const uint8_t  
                                              uuid[16])
```

Set UUID in the nboot context.

This function sets the UUID (Universally Unique Identifier) in the nboot context structure. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `uuid` – Pointer to 16-byte UUID array to be set in the context.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

```
nboot_status_protected_t NBOOT_Sb4LoadManifest(nboot_context_t *context, const uint32_t  
                                              *manifest,  
                                              nboot_sb4_load_manifest_parms_t *parms)
```

Load NBOOT SB4 manifest.

This function loads and processes an NBOOT SB4 manifest. The manifest contains metadata and configuration information for the SB4 file processing. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `manifest` – Pointer to the SB4 manifest data.
- `parms` – Pointer to a data structure in trusted memory, holding input parameters for the algorithm. The data structure shall be correctly filled before the function call.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

```
nboot_status_protected_t NBOOT_Sb4LoadBlock(nboot_context_t *context, uint32_t *block)
```

Load NBOOT SB4 block.

This function loads and processes an NBOOT SB4 block. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `block` – Pointer to the SB4 block data to be loaded.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

```
nboot_status_protected_t NBOOT_SB4CheckAuthenticityAndCompleteness(nboot_context_t  
                                                                    *context, const  
                                                                    uint32_t *address,  
                                                                    nboot_sb4_load_manifest_parms_t  
                                                                    *parms)
```

Check authenticity and completeness of NBOOT SB4 file.

This function verifies the authenticity and completeness of an NBOOT SB4 file. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `address` – Pointer to the SB4 file data in memory.
- `parms` – Pointer to a data structure in trusted memory, holding input parameters for the algorithm. The data structure shall be correctly filled before the function call.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

```
nboot_status_protected_t NBOOT_ImgAuthenticate(nboot_context_t *context, const uint8_t
imageStartAddress[], nboot_bool_t
*isSignatureVerified,
nboot_img_auth_parms_t *parms)
```

Authenticate NBOOT image.

This function authenticates an NBOOT image by verifying its signature and integrity. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- `context` – Pointer to `nboot_context_t` structure.
- `imageStartAddress` – Pointer to the start address of the image data to be authenticated.
- `isSignatureVerified` – Pointer to a boolean flag that will be set to indicate if the signature was verified.
- `parms` – Pointer to a data structure in trusted memory, holding input parameters for the algorithm. The data structure shall be correctly filled before the function call.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

```
nboot_status_protected_t NBOOT_MemCryptEnableEncryptForAddressRange(nboot_context_t
*context,
nboot_mem_crypt_region_t
regionNumber,
nboot_mem_crypt_region_config_t
*regionConfig,
nboot_mem_crypt_iped_mode_select_t
ipedModeSelect)
```

Enable memory encryption for address range.

This function enables memory encryption for a specified address range using the NBOOT memory cryptography features. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- context – Pointer to `nboot_context_t` structure.
- regionNumber – The memory region number to configure.
- regionConfig – Pointer to the region configuration structure.
- ipedModeSelect – IPED mode selection parameter.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

`nboot_status_protected_t` NBOOT_MemCryptRangeChecker(*nboot_context_t* *context, *nboot_mem_crypt_operation_t* operation, *uint32_t* address, *uint32_t* length, *uint8_t* *flags, *uint32_t* *npx_iv_erase_cntr, *uint32_t* *iped_iv_erase_cntr, *uint32_t* npx_erase_check_en)

Memory cryptography range checker.

This function performs range checking for memory cryptography operations. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- context – Pointer to `nboot_context_t` structure.
- operation – The memory cryptography operation type.
- address – The start address to check.
- length – The length of the memory range to check.
- flags – Pointer to flags output parameter.
- `npx_iv_erase_cntr` – Pointer to NPX IV erase counter output parameter.
- `iped_iv_erase_cntr` – Pointer to IPED IV erase counter output parameter.
- `npx_erase_check_en` – NPX erase check enable parameter.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

`nboot_status_protected_t` NBOOT_BackgroundHashEnable(*nboot_context_t* *context, *uint32_t* hashDmaChannel)

Enable background hash.

This function enables background hash functionality using the specified DMA channel. The NBOOT context has to be initialized by the function `nboot_context_init` before calling this function.

Parameters

- context – Pointer to `nboot_context_t` structure.
- hashDmaChannel – The DMA channel to use for background hash operations.

Return values

- `kStatus_NBOOT_Success` – Operation successfully finished
- `kStatus_NBOOT_Fail` – Error occurred during operation

NBOOT_RKTH_SIZE_IN_WORDS

struct __nboot_secure_counter

#include <fsl_nboot.h> Data structure holding secure counter value used by nboot library.

struct nboot_trng_cfg_load

#include <fsl_nboot.h>

struct nboot_context_struct

#include <fsl_nboot.h>

struct nboot_rot_auth_parms__

#include <fsl_nboot.h> NBOOT type for the root of trust parameters.

This type defines the NBOOT root of trust parameters

struct nboot_img_auth_parms_struct

#include <fsl_nboot.h> Data structure holding input arguments to POR secure boot (authentication) algorithm. Shall be read from SoC trusted NVM or SoC fuses.

struct nboot_sb4_load_manifest_parms_struct

#include <fsl_nboot.h> manifest loading parameters

This type defines the NBOOT SB4 manifest loading parameters

struct nboot_npx_region_config_struct

#include <fsl_nboot.h>

struct nboot_iped_region_config_struct

#include <fsl_nboot.h>

union nboot_mem_crypt_region_config_union

#include <fsl_nboot.h>

2.61 OSTIMER: OS Event Timer Driver

void OSTIMER_Init(OSTIMER_Type *base)

Initializes an OSTIMER by turning its bus clock on.

void OSTIMER_Deinit(OSTIMER_Type *base)

Deinitializes a OSTIMER instance.

This function shuts down OSTIMER bus clock

Parameters

- base – OSTIMER peripheral base address.

uint64_t OSTIMER_GrayToDecimal(uint64_t gray)

Translate the value from gray-code to decimal.

Parameters

- gray – The gray value input.

Returns

The decimal value.

static inline uint64_t OSTIMER_DecimalToGray(uint64_t dec)

Translate the value from decimal to gray-code.

Parameters

- dec – The decimal value.

Returns

The gray code of the input value.

`uint32_t OSTIMER_GetStatusFlags(OSTIMER_Type *base)`

Get OSTIMER status Flags.

This returns the status flag. Currently, only match interrupt flag can be got.

Parameters

- `base` – OSTIMER peripheral base address.

Returns

status register value

`void OSTIMER_ClearStatusFlags(OSTIMER_Type *base, uint32_t mask)`

Clear Status Interrupt Flags.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

- `base` – OSTIMER peripheral base address.
- `mask` – Clear bit mask.

Returns

none

`status_t OSTIMER_SetMatchRawValue(OSTIMER_Type *base, uint64_t count, ostimer_callback_t cb)`

Set the match raw value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format may be gray-code, if so, please using `OSTIMER_SetMatchValue()`.

Parameters

- `base` – OSTIMER peripheral base address.
- `count` – OSTIMER timer match value.(Value may be gray-code format)
- `cb` – OSTIMER callback (can be left as NULL if none, otherwise should be a `void func(void)`).

Return values

`kStatus_Success` -- Set match raw value and enable interrupt Successfully.

`status_t OSTIMER_SetMatchValue(OSTIMER_Type *base, uint64_t count, ostimer_callback_t cb)`

Set the match value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

Parameters

- `base` – OSTIMER peripheral base address.
- `count` – OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code in API if the IP counter is gray encoded.)
- `cb` – OSTIMER callback (can be left as NULL if none, otherwise should be a `void func(void)`).

Return values

`kStatus_Success` -- Set match raw value and enable interrupt Successfully.

```
static inline void OSTIMER_SetMatchRegister(OSTIMER_Type *base, uint64_t value)
```

Set value to OSTIMER MATCH register directly.

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER_DecimalToGray could convert decimal value to gray code.

Parameters

- base – OSTIMER peripheral base address.
- value – OSTIMER timer match value (Value is gray-code format).

```
static inline uint64_t OSTIMER_GetMatchRegister(OSTIMER_Type *base)
```

Get the match value from OSTIMER.

This function will get the match value from OSTIMER. The value of timer match is gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register, data format is gray code.

```
static inline uint64_t OSTIMER_GetMatchValue(OSTIMER_Type *base)
```

Get the match value from OSTIMER.

This function will get a match value from OSTIMER.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register.

```
static inline void OSTIMER_EnableMatchInterrupt(OSTIMER_Type *base)
```

Enable the OSTIMER counter match interrupt.

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

```
static inline void OSTIMER_DisableMatchInterrupt(OSTIMER_Type *base)
```

Disable the OSTIMER counter match interrupt.

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

```
static inline uint64_t OSTIMER_GetCurrentTimerRawValue(OSTIMER_Type *base)
```

Get current timer raw count value from OSTIMER.

This function will get the timer count value from OS timer register. The raw value of timer count may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, may be gray code format.

uint64_t OSTIMER_GetCurrentTimerValue(OSTIMER_Type *base)

Get current timer count value from OSTIMER.

This function will get a decimal timer count value. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formatted to decimal value.

static inline uint64_t OSTIMER_GetCaptureRawValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a captured value from OSTIMER. The Raw value of timer capture may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of capture register, data format may be gray code.

uint64_t OSTIMER_GetCaptureValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a capture decimal-value from OSTIMER. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of capture register, data format is decimal.

void OSTIMER_HandleIRQ(OSTIMER_Type *base, *ostimer_callback_t* cb)

OS timer interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – OS timer peripheral base address.
- cb – callback scheduled for this instance of OS timer

Returns

none

FSL_OSTIMER_DRIVER_VERSION

OSTIMER driver version.

enum *_ostimer_flags*

OSTIMER status flags.

Values:

enumerator kOSTIMER_MatchInterruptFlag

Match interrupt flag bit, sets if the match value was reached.

typedef void (**ostimer_callback_t*)(void)

ostimer callback function.

2.62 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.

2.63 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 kCTIMER2_RST_SHIFT_RSTn
CTIMER2 reset control

enumerator kCTIMER3_RST_SHIFT_RSTn
CTIMER3 reset control

enumerator kCTIMER4_RST_SHIFT_RSTn
CTIMER4 reset control

enumerator kUTICK0_RST_SHIFT_RSTn
UTICK0 reset control

enumerator kDMA0_RST_SHIFT_RSTn
DMA0 reset control

enumerator kDMA1_RST_SHIFT_RSTn
DMA1 reset control

enumerator kAOI0_RST_SHIFT_RSTn
AOI0 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 kFLEXIO0_RST_SHIFT_RSTn
FLEXIO0 reset control

enumerator kLPI2C0_RST_SHIFT_RSTn
LPI2C0 reset control

enumerator kLPI2C1_RST_SHIFT_RSTn
LPI2C1 reset control

enumerator kLPI2C2_RST_SHIFT_RSTn
LPI2C2 reset control

enumerator kLPI2C3_RST_SHIFT_RSTn
LPI2C3 reset control

enumerator kLPI2C4_RST_SHIFT_RSTn
LPI2C4 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 kLPUART4_RST_SHIFT_RSTn
LPUART4 reset control

enumerator kLPUART5_RST_SHIFT_RSTn
LPUART5 reset control

enumerator kOSTIMER0_RST_SHIFT_RSTn
OSTIMER0 reset control

enumerator kLPSP10_RST_SHIFT_RSTn
LPSP10 reset control

enumerator kLPSP11_RST_SHIFT_RSTn
LPSP11 reset control

enumerator kLPSP12_RST_SHIFT_RSTn
LPSP12 reset control

enumerator kLPSP13_RST_SHIFT_RSTn
LPSP13 reset control

enumerator kLPSP14_RST_SHIFT_RSTn
LPSP14 reset control

enumerator kLPSP15_RST_SHIFT_RSTn
LPSP15 reset control

enumerator kPORT0_RST_SHIFT_RSTn
PORT0 reset control

enumerator kPORT1_RST_SHIFT_RSTn
PORT1 reset control

enumerator kPORT2_RST_SHIFT_RSTn
PORT2 reset control

enumerator kPORT3_RST_SHIFT_RSTn
PORT3 reset control

enumerator kPORT4_RST_SHIFT_RSTn
PORT4 reset control

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control

enumerator kADC1_RST_SHIFT_RSTn
ADC1 reset control

enumerator kDAC0_RST_SHIFT_RSTn
DAC0 reset control

enumerator kDAC1_RST_SHIFT_RSTn
DAC1 reset control

enumerator kVREF0_RST_SHIFT_RSTn
VREF0 reset control

enumerator kI3C0_RST_SHIFT_RSTn
I3C0 reset control

enumerator kI3C1_RST_SHIFT_RSTn
I3C1 reset control

enumerator kI3C2_RST_SHIFT_RSTn
I3C2 reset control

enumerator kI3C3_RST_SHIFT_RSTn
I3C3 reset control

enumerator kFLEXCAN0_RST_SHIFT_RSTn
FLEXCAN0 reset control

enumerator kFLEXCAN1_RST_SHIFT_RSTn
FLEXCAN1 reset control

enumerator kENET0_RST_SHIFT_RSTn
ENET0 reset control

enumerator kT1S0_RST_SHIFT_RSTn
T1S0 reset control

enumerator kFLEXSPI0_RST_SHIFT_RSTn
FLEXSPI0 reset control

enumerator kSPIFILTER0_RST_SHIFT_RSTn
SPIFILTER0 reset control

enumerator kESPI0_RST_SHIFT_RSTn
ESPI0 reset control

enumerator kUSB1_RST_SHIFT_RSTn
USB1 reset control

enumerator kUSB1_PHY_RST_SHIFT_RSTn
USB1_PHY reset control

enumerator kEWM0_RST_SHIFT_RSTn
EWM0 reset control

enumerator kGPIO0_RST_SHIFT_RSTn
GPIO0 reset control

enumerator kGPIO1_RST_SHIFT_RSTn
GPIO1 reset control

enumerator kGPIO2_RST_SHIFT_RSTn
GPIO2 reset control

enumerator kGPIO3_RST_SHIFT_RSTn
GPIO3 reset control

enumerator kGPIO4_RST_SHIFT_RSTn
GPIO4 reset control

enumerator kSMART_DMA_RST_SHIFT_RSTn
SMARTDMA0 reset control

enumerator kGLIKEY0_RST_SHIFT_RSTn
GLIKEY0 reset control

enumerator kPKC0_RST_SHIFT_RSTn
PKC0 reset control

enumerator kTRNG0_RST_SHIFT_RSTn
TRNG0 reset control

enumerator kDGDET0_RST_SHIFT_RSTn

DGDET0 reset control

enumerator kATX0_RST_SHIFT_RSTn

ATX0 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

ADC_RSTS

Reset bits for ADC peripheral.

Array initializers with peripheral reset bits

ADC_RSTS_N

AOI_RSTS

Reset bits for AOI peripheral.

AOI_RSTS_N

ATX_RSTS

Reset bits for ATX peripheral.

ATX_RSTS_N

CRC_RSTS

Reset bits for CRC peripheral.

CRC_RSTS_N

CTIMER_RSTS

Reset bits for CTIMER peripheral.

CTIMER_RSTS_N

DAC_RSTS

Reset bits for DAC peripheral.

DAC_RSTS_N

DGDET_RSTS

Reset bits for DGDET peripheral.

DGDET_RSTS_N

DMA_RSTS

Reset bits for DMA peripheral.

DMA_RSTS_N

EIM_RSTS

Reset bits for EIM peripheral.

EIM_RSTS_N

ENET_RSTS

Reset bits for ENET peripheral.

ENET_RSTS_N

ERM_RSTS

Reset bits for ERM peripheral.

ERM_RSTS_N

ESPI_RSTS

Reset bits for ESPI peripheral.

ESPI_RSTS_N

EWM_RSTS

Reset bits for EWM peripheral.

EWM_RSTS_N

FLEXCAN_RSTS

Reset bits for FLEXCAN peripheral.

FLEXCAN_RSTS_N

FLEXIO_RSTS

Reset bits for FLEXIO peripheral.

FLEXIO_RSTS_N

FLEXSPI_RSTS

Reset bits for FLEXSPI peripheral.

FLEXSPI_RSTS_N

FREQME_RSTS

Reset bits for FREQME peripheral.

FREQME_RSTS_N

GLIKEY_RSTS

Reset bits for GLIKEY peripheral.

GLIKEY_RSTS_N

GPIO_RSTS

Reset bits for GPIO peripheral.

GPIO_RSTS_N

I3C_RSTS

Reset bits for I3C peripheral.

I3C_RSTS_N

INPUTMUX_RSTS

Reset bits for INPUTMUX peripheral.

INPUTMUX_RSTS_N

LPI2C_RSTS

Reset bits for LPI2C peripheral.

LPI2C_RSTS_N

LPSPI_RSTS

Reset bits for LPSPI peripheral.

LPSPI_RSTS_N

LPUART_RSTS

Reset bits for LPUART peripheral.

LPUART_RSTS_N

OSTIMER_RSTS

Reset bits for OSTIMER peripheral.

OSTIMER_RSTS_N

PKC_RSTS

Reset bits for PKC peripheral.

PKC_RSTS_N

PORT_RSTS

Reset bits for PORT peripheral.

PORT_RSTS_N

SMARTDMA_RSTS

Reset bits for SMARTDMA peripheral.

SMARTDMA_RSTS_N

SPIFILTER_RSTS

Reset bits for SPIFILTER peripheral.

SPIFILTER_RSTS_N

T1S_RSTS

Reset bits for T1S peripheral.

T1S_RSTS_N

TENBASET_PHY_RST

TENBASET_PHY_RST_N

TRNG_RSTS

Reset bits for TRNG peripheral.

TRNG_RSTS_N

USB_RSTS

Reset bits for USB peripheral.

USB_RSTS_N

USB_PHY_RSTS

Reset bits for USB PHY peripheral.

USB_PHY_RSTS_N

UTICK_RSTS

Reset bits for UTICK peripheral.

UTICK_RSTS_N

VREF_RSTS

Reset bits for VREF peripheral.

VREF_RSTS_N

2.64 ROMAPI Driver

2.65 Runbootloader

void ROMAPI_RunBootloader(void *arg)

Run the Bootloader API to force into the ISP mode base on the user arg.

Parameters

- arg – Indicates API prototype fields definition. Refer to the above user_app_boot_invoke_option_t structure

FSL_ROMAPI_RUNBOOTLOADER_DRIVER_VERSION

ROMAPI_RUNBOOTLOADER driver version 2.0.0.

uint32_t reserved

```
uint32_t recovery__boot_cfg0
uint32_t recovery__boot_cfg1
uint32_t boot__image_index
uint32_t instance
uint32_t boot__interface
uint32_t mode
uint32_t tag
struct user_app_boot_invoke_option_t B
uint32_t U
union user_app_boot_invoke_option_t option
struct user_app_boot_invoke_option_t
    #include <fsl_runbootloader.h>
union option
```

Public Members

```
struct user_app_boot_invoke_option_t B
uint32_t U
struct B
```

2.66 RUNBOOTLOADER Driver

2.67 MCXN SMARTDMA Firmware

```
enum __smartdma_display_api
    The API index when using s_smartdmaDisplayFirmware.
    Values:
    enumerator kSMARTDMA_FlexIO_DMA

enum __smartdma_camera_api
    The API index when using s_smartdmaCameraFirmware.
    Values:
    enumerator kSMARTDMA_FlexIO_CameraWholeFrame
    enumerator kSMARTDMA_FlexIO_CameraDiv16Frame
        Deprecated. Use kSMARTDMA_CameraWholeFrameQVGA instead.
    enumerator kSMARTDMA_CameraWholeFrameQVGA
        Deprecated. Use kSMARTDMA_CameraDiv16FrameQVGA instead.
        Save whole frame of QVGA(320x240) to buffer in each interrupt in RGB565 format.
```


enumerator kSMARTDMA_CameraDiv16FrameQVGA

Save 1/16 frame of QVGA(320x240) to buffer in each interrupt in RGB565 format, takes 16 interrupts to get the whole frame.

enumerator kSMARTDMA_CameraWholeFrame480_320

Save whole frame of 480x320 to buffer in each interrupt in RGB565 format.

enumerator kSMARTDMA_CameraDiv4FrameQVGAGrayScale

Save 1/4 frame of QVGA(320x240) to buffer in each interrupt in grayscale format, takes 4 interrupts to get the whole frame.

enumerator kSMARTDMA_CameraDiv16FrameQVGAGrayScale

Save 1/16 frame of QVGA(320x240) to buffer in each interrupt in grayscale format, takes 16 interrupts to get the whole frame.

enumerator kSMARTDMA_CameraDiv16Frame384_384

Save 1/16 frame of 384x384 to buffer in each interrupt in grayscale format, takes 16 interrupts to get the whole frame.

enumerator kSMARTDMA_CameraWholeFrame320_480

Save whole frame of 320x480 to buffer in each interrupt in RGB565 format.

enum _smartdma_keyscan_api

The API index when using s_smartdmaKeyscanFirmware.

Values:

enumerator kSMARTDMA_Keyscan_4x4

Using SmartDma to control GPIO.

typedef struct _smartdma_flexio_mculcd_param smartdma_flexio_mculcd_param_t

Parameter for FlexIO MCULCD.

typedef struct _smartdma_camera_param smartdma_camera_param_t

Parameter for camera.

typedef struct _smartdma_keyscan_4x4_param smartdma_keyscan_4x4_param_t

Parameter for keyscan 4x4.

const uint8_t s_smartdmaDisplayFirmware[]

The firmware used for display.

const uint32_t s_smartdmaDisplayFirmwareSize

Size of s_smartdmaDisplayFirmware.

const uint8_t s_smartdmaCameraFirmware[]

The firmware used for camera.

const uint32_t s_smartdmaCameraFirmwareSize

Size of s_smartdmacameraFirmware.

const uint8_t s_smartdmaKeyscanFirmware[]

The firmware used for keyscan.

const uint32_t s_smartdmaKeyscanFirmwareSize

Size of s_smartdmaKeyscanFirmware.

FSL_SMARTDMA_MCXA_DRIVER_VERSION

SMARTDMA driver version.

SMARTDMA_DISPLAY_MEM_ADDR

The s_smartdmaDisplayFirmware firmware memory address.

SMARTDMA_DISPLAY_FIRMWARE_SIZE

Size of s_smartdmaDisplayFirmware.

SMARTDMA_CAMERA_MEM_ADDR

The s_smartdmaCameraFirmware firmware memory address.

SMARTDMA_CAMERA_FIRMWARE_SIZE

Size of s_smartdmacameraFirmware.

SMARTDMA_KEYSCAN_MEM_ADDR

The s_smartdmaKeyscanFirmware firmware memory address.

SMARTDMA_KEYSCAN_FIRMWARE_SIZE

Size of s_smartdmaKeyscanFirmware.

uint32_t *p_buffer

uint32_t buffersize

uint32_t *smartdma_stack

uint32_t *smartdma_stack

Stack used by SMARTDMA, shall be at least 64 bytes.

uint32_t *p_buffer

Buffer to store the received camera data.

uint32_t *p_stripe_index

Pointer to stripe index. Used when only partial frame is received per interrupt.

uint32_t *p_buffer_ping_pong

Buffer to store the 2nd stripe of camera data. Used when only partial frame is received per interrupt.

smartdma_flexio_mculcd_param_t flexioMcuLcdParam

Parameter for flexio MCULCD.

smartdma_camera_param_t cameraParam

Parameter for camera.

uint32_t *smartdma_stack

Stack used by SMARTDMA, shall be at least 64 bytes.

uint32_t *p_gpio_reg

Buffer to provide GPIO register for COL1, COL2, COL3, COL4, ROW1, ROW2, ROW3, ROW4.

uint32_t *p_keyvalue

Buffer to store key value.

uint32_t *p_keycan_interval

Delay how many system clock cycles.

struct _smartdma_flexio_mculcd_param

#include <fsl_smartdma_fw.h> Parameter for FlexIO MCULCD.

struct _smartdma_camera_param

#include <fsl_smartdma_fw.h> Parameter for camera.

union smartdma_param_t

#include <fsl_smartdma_fw.h> Parameter for all supported APIs.

struct _smartdma_keyscan_4x4_param

#include <fsl_smartdma_fw.h> Parameter for keyscan 4x4.

2.68 TENBASET_PHY: 10BASE-T1S Digital PHY Driver

enum _tenbaset_phy_mode

TENBASET_PHY operating modes.

Values:

enumerator kTENBASET_PHY_ModeNone

No mode command

enumerator kTENBASET_PHY_ModeLinkDown

Link down mode

enumerator kTENBASET_PHY_ModeLinkUp

Link up mode

enumerator kTENBASET_PHY_ModeTxcCfg

Transceiver configuration mode

enumerator kTENBASET_PHY_ModeLowPower

Low power mode

enumerator kTENBASET_PHY_ModeTxcBist

Transceiver BIST mode

enum _tenbaset_phy_mode_status

TENBASET_PHY mode status.

Values:

enumerator kTENBASET_PHY_StatusPorst

Power-on reset state

enumerator kTENBASET_PHY_StatusWaitInit

Wait for initialization

enumerator kTENBASET_PHY_StatusLinkDown

Link down state

enumerator kTENBASET_PHY_StatusLinkUp

Link up state

enumerator kTENBASET_PHY_StatusWaitCmdLp

Wait for low power command

enumerator kTENBASET_PHY_StatusLowPower

Low power state

enumerator kTENBASET_PHY_StatusWaitCmdCfg

Wait for configuration command

enumerator kTENBASET_PHY_StatusTxcCfg

Transceiver configuration state

enumerator kTENBASET_PHY_StatusTxcBist

Transceiver BIST state

enumerator kTENBASET_PHY_StatusWaitSilentLp

Wait silent for low power

enumerator kTENBASET_PHY_StatusWaitSilentCfg

Wait silent for configuration

enum _tenbaset_phy_plca_diag_flags

TENBASET_PHY PLCA diagnostic flags.

Values:

enumerator kTENBASET_PHY_RxInTo

Reception occurred within local transmit opportunity

enumerator kTENBASET_PHY_UnexpectedBeacon

Reception of unexpected BEACON by coordinator

enumerator kTENBASET_PHY_BeaconBeforeTo

4 consecutive BEACONs received before TO reached

enumerator kTENBASET_PHY_UndefinedState

PLCA state-machines in undefined state

enumerator kTENBASET_PHY_NoRxBeacon

No beacon received by coordinator while transmitted

enumerator kTENBASET_PHY_LateBeacon

Reception of late BEACON by follower node

enumerator kTENBASET_PHY_EarlyBeacon

Reception of early BEACON by follower node

enum _tenbaset_phy_wup_error_code

TENBASET_PHY WUP error codes.

Values:

enumerator kTENBASET_PHY_WupErrorNone

WUP transmission successful

enumerator kTENBASET_PHY_WupErrorSwAbort

WUP sequencer aborted by software

enumerator kTENBASET_PHY_WupErrorModeChg

WUP sequencer aborted by invalid mode

enumerator kTENBASET_PHY_WupErrorToWm

WUP sequencer aborted by time-out in WAITMODE state

enumerator kTENBASET_PHY_WupErrorToPd

WUP sequencer aborted by time-out in PENDING state

enumerator kTENBASET_PHY_WupErrorToWp

WUP sequencer aborted by time-out in PENDING state

enumerator kTENBASET_PHY_WupErrorToWa

WUP sequencer aborted by time-out in PENDING state

enumerator kTENBASET_PHY_WupErrorCore

WUP sequencer aborted by core

enum _tenbaset_phy_wup_status

TENBASET_PHY WUP sequencer status.

Values:

enumerator kTENBASET_PHY_WupStatusIdle

WUP sequencer is idle

enumerator kTENBASET_PHY_WupStatusWaitLink
WUP sequencer is waiting for LINKUP mode

enumerator kTENBASET_PHY_WupStatusPending
WUP sequencer is waiting for transmission slot

enum __tenbaset_phy_txc_diag_flags
TENBASET_PHY transceiver diagnostic flags.

Values:

enumerator kTENBASET_PHY_LowPowerFail
Failure of ED/RX pin in LOWPOWER mode

enumerator kTENBASET_PHY_EdHighFail
Stuck HIGH of the ED pin

enumerator kTENBASET_PHY_EdLowFail
Stuck LOW of the ED pin

enumerator kTENBASET_PHY_RxHighFail
Stuck HIGH of the RX pin

enumerator kTENBASET_PHY_RxLowFail
Stuck LOW of the RX pin

enum __tenbaset_phy_interrupt_flags
TENBASET_PHY interrupt flags.

Values:

enumerator kTENBASET_PHY_PhysicalCollisionFlag
Physical collision interrupt

enumerator kTENBASET_PHY_PLCARecoveryFlag
PLCA recovery interrupt

enumerator kTENBASET_PHY_PLCAStatusFlag
PLCA status interrupt

enumerator kTENBASET_PHY_ModeStatusFlag
Mode status interrupt

enumerator kTENBASET_PHY_InvalidAPBFlag
Invalid APB interrupt

enumerator kTENBASET_PHY_LocalJabberFlag
Local jabber interrupt

enumerator kTENBASET_PHY_RemoteJabberFlag
Remote jabber interrupt

enumerator kTENBASET_PHY_PinFaultFlag
Pin fault interrupt

enumerator kTENBASET_PHY_PLCADiagFlag
PLCA diagnostics interrupt

enumerator kTENBASET_PHY_SMIAccessFlag
SMI access interrupt

enumerator kTENBASET_PHY_LocalWakeFlag
Local wake interrupt

enumerator kTENBASET_PHY_SuspendDetectFlag

Suspend detect interrupt

enumerator kTENBASET_PHY_WUTDetectFlag

WUT detect interrupt

enumerator kTENBASET_PHY_WUPDoneFlag

WUP done interrupt

enumerator kTENBASET_PHY_APBParityFlag

APB parity interrupt

enum _tenbaset_phy_smi_operation

SMI operation types.

Values:

enumerator kTENBASET_PHY_SMIWriteWithCompError

Write frame with C22 compliance error

enumerator kTENBASET_PHY_SMIWrite

Standard write frame operation

enumerator kTENBASET_PHY_SMIRead

Standard read frame operation

enumerator kTENBASET_PHY_SMIReadWithCompError

Read frame with C22 compliance error

enum _tenbaset_phy_smi_hold_time

SMI MDIO hold time options.

Values:

enumerator kTENBASET_PHY_SMIHold1Cycle

1 PCLK cycle

enumerator kTENBASET_PHY_SMIHold2Cycles

2 PCLK cycles

enumerator kTENBASET_PHY_SMIHold3Cycles

3 PCLK cycles

enumerator kTENBASET_PHY_SMIHold4Cycles

4 PCLK cycles

enumerator kTENBASET_PHY_SMIHold5Cycles

5 PCLK cycles

enumerator kTENBASET_PHY_SMIHold6Cycles

6 PCLK cycles

enumerator kTENBASET_PHY_SMIHold7Cycles

7 PCLK cycles

enumerator kTENBASET_PHY_SMIHold8Cycles

8 PCLK cycles

enum _tenbaset_phy_smi_error_code

SMI error codes.

Values:

```
enumerator kTENBASET_PHY_SMIErrorNone
    SMI access successful

enumerator kTENBASET_PHY_SMIErrorNonTC14
    Non-compliance with TC14 detected

enumerator kTENBASET_PHY_SMIErrorDisabled
    SMI access triggered while disabled

enumerator kTENBASET_PHY_SMIErrorAborted
    SMI access got aborted

enumerator kTENBASET_PHY_SMIErrorInProgress
    SMI access triggered while in progress

typedef enum _tenbaset_phy_mode tenbaset_phy_mode_t
    TENBASET_PHY operating modes.

typedef enum _tenbaset_phy_mode_status tenbaset_phy_mode_status_t
    TENBASET_PHY mode status.

typedef struct _tenbaset_phy_plca_config tenbaset_phy_plca_config_t
    TENBASET_PHY PLCA configuration structure.

typedef struct _tenbaset_phy_pma_config tenbaset_phy_pma_config_t
    PMA configuration structure.

typedef struct _tenbaset_phy_pcs_config tenbaset_phy_pcs_config_t
    PCS configuration structure.

typedef struct _tenbaset_phy_wake_config tenbaset_phy_wake_config_t
    TENBASET_PHY wake configuration structure.

typedef enum _tenbaset_phy_wup_error_code tenbaset_phy_wup_error_code_t
    TENBASET_PHY WUP error codes.

typedef enum _tenbaset_phy_wup_status tenbaset_phy_wup_status_t
    TENBASET_PHY WUP sequencer status.

typedef struct _tenbaset_phy_wup_status_info tenbaset_phy_wup_status_info_t
    TENBASET_PHY WUP status structure.

typedef struct _tenbaset_phy_version_info tenbaset_phy_version_info_t
    TENBASET_PHY version information structure.

typedef enum _tenbaset_phy_smi_operation tenbaset_phy_smi_operation_t
    SMI operation types.

typedef enum _tenbaset_phy_smi_hold_time tenbaset_phy_smi_hold_time_t
    SMI MDIO hold time options.

typedef struct _tenbaset_phy_smi_config tenbaset_phy_smi_config_t
    SMI configuration structure.

typedef enum _tenbaset_phy_smi_error_code tenbaset_phy_smi_error_code_t
    SMI error codes.

typedef void (*tenbaset_phy_callback_t)(TENBASET_PHY_Type *base, uint16_t flags, void
*userData)
    TENBASET_PHY callback function type.

typedef struct _tenbaset_phy_config tenbaset_phy_config_t
    TENBASET_PHY configuration structure.
```

`typedef struct tenbaset_phy_handle tenbaset_phy_handle_t`
TENBASET_PHY handle structure.

`FSL_TENBASET_PHY_DRIVER_VERSION`
TENBASET_PHY driver version.

`TENBASET_PHY_PLCADIAG1_FLAG(flag)`

`TENBASET_PHY_PLCADIAG2_FLAG(flag)`

`TENBASET_PHY_PLCADIAG1_FLAG_TO_REG(flag)`

`TENBASET_PHY_PLCADIAG2_FLAG_TO_REG(flag)`

`void TENBASET_PHY_GetDefaultConfig(tenbaset_phy_config_t *config)`
Gets the default configuration structure.

This function initializes the TENBASET_PHY configuration structure to default values. The default values are:

```
config->plcaConfig.nodeId      = 0xFFU;  
config->plcaConfig.nodeCount   = 0x08U;  
config->plcaConfig.toTimer     = 0x20U;  
config->plcaConfig.burstTimer  = 0x80U;  
config->plcaConfig.maxBurstCount = 0U;  
config->plcaConfig.enable      = false;  
config->wakeConfig.suspendWakeEnable = true;  
config->wakeConfig.remoteWakeEnable  = true;  
config->wakeConfig.localWakeEnable   = true;  
config->wakeConfig.remoteWakeForwardEnable = true;  
config->wakeConfig.localWakeWupEnable = true;  
config->wakeConfig.wupTimeout        = 0xFFU;  
config->interruptMask = 0U;
```

Parameters

- config – Pointer to the TENBASET_PHY configuration structure.

`void TENBASET_PHY_CreateHandle(TENBASET_PHY_Type *base, tenbaset_phy_handle_t *handle, tenbaset_phy_callback_t callback, void *userData)`

Initializes driver handle for TENBASET_PHY instance.

This function initializes the handle with the provided data.

Parameters

- base – TENBASET_PHY peripheral base address.
- handle – Pointer to the TENBASET_PHY handle structure.
- callback – Pointer to the callback function called from ISR.
- userData – Pointer to the user data (passed to the callback).

`status_t TENBASET_PHY_Init(TENBASET_PHY_Type *base, const tenbaset_phy_config_t *config, tenbaset_phy_handle_t *handle)`

Initializes the TENBASET_PHY peripheral.

This function initializes the TENBASET_PHY peripheral with the provided configuration.

Parameters

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

- `handle` – Pointer to the `TENBASET_PHY` handle structure.

Return values

- `kStatus_Success` – Initialization successful.
- `kStatus_Timeout` – Initialization timeout.
- `kStatus_Fail` – PLCA configuration error.

`void TENBASET_PHY_Deinit(TENBASET_PHY_Type *base, tenbaset_phy_handle_t *handle)`
Deinitializes the `TENBASET_PHY` peripheral.

Parameters

- `base` – `TENBASET_PHY` peripheral base address.
- `handle` – Pointer to the `TENBASET_PHY` handle structure.

`status_t TENBASET_PHY_SetMode(TENBASET_PHY_Type *base, tenbaset_phy_mode_t mode)`
Sets the `TENBASET_PHY` operating mode.

Parameters

- `base` – `TENBASET_PHY` peripheral base address.
- `mode` – Target operating mode.

Return values

- `kStatus_Success` – Mode set successfully.
- `kStatus_Timeout` – Mode transition timeout.
- `kStatus_InvalidArgument` – Invalid mode transition.

`tenbaset_phy_mode_status_t TENBASET_PHY_GetModeStatus(TENBASET_PHY_Type *base)`
Gets the current `TENBASET_PHY` mode status.

Parameters

- `base` – `TENBASET_PHY` peripheral base address.

Returns

Current mode status.

`bool TENBASET_PHY_IsLinkUp(TENBASET_PHY_Type *base)`
Checks if the link is up.

Parameters

- `base` – `TENBASET_PHY` peripheral base address.

Returns

true if link is up, false otherwise.

`void TENBASET_PHY_GetPLCAConfig(TENBASET_PHY_Type *base,
tenbaset_phy_plca_config_t *config)`

Retrieves the current Physical Layer Collision Avoidance (PLCA) configuration.

This function reads the current PLCA settings from the registers, including node ID, node count, timers, and PLCA enable status.

Parameters

- `base` – `TENBASET_PHY` peripheral base address
- `config` – Pointer to store the current PLCA configuration

status_t TENBASET_PHY_SetPLCAConfig(TENBASET_PHY_Type *base, const
tenbaset_phy_plca_config_t *config)

Configures PLCA settings.

Parameters

- base – TENBASET_PHY peripheral base address.
- config – Pointer to PLCA configuration structure.

Return values

- kStatus_Success – PLCA configured successfully.
- kStatus_Fail – PLCA configuration error.

bool TENBASET_PHY_GetPLCAStatus(TENBASET_PHY_Type *base)

Gets PLCA status.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

PLCA node status.

uint32_t TENBASET_PHY_GetPLCADiagnosticFlags(TENBASET_PHY_Type *base)

Gets PLCA diagnostic flags.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

diagnostic flags (combination of *_tenbaset_phy_plca_diag_flags*).

void TENBASET_PHY_ClearPLCADiagnosticFlags(TENBASET_PHY_Type *base, *uint32_t*
diagFlags)

Clears PLCA diagnostic flags.

Parameters

- base – TENBASET_PHY peripheral base address.
- diagFlags – Flags to clear (combination of *_tenbaset_phy_plca_diag_flags*).

uint16_t TENBASET_PHY_GetBeaconCounter(TENBASET_PHY_Type *base)

Gets PLCA beacon counter.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

RX beacon counter value.

void TENBASET_PHY_ClearBeaconCounter(TENBASET_PHY_Type *base)

Clears PLCA beacon counter.

Parameters

- base – TENBASET_PHY peripheral base address.

uint16_t TENBASET_PHY_GetTransmitOpportunityCounter(TENBASET_PHY_Type *base)

Gets PLCA transmit opportunity counter.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

Transmit opportunity counter value.

```
void TENBASET_PHY_ClearTransmitOpportunityCounter(TENBASET_PHY_Type *base)
```

Clears PLCA transmit opportunity counter.

Parameters

- base – TENBASET_PHY peripheral base address.

```
void TENBASET_PHY_GetPMAConfig(TENBASET_PHY_Type *base, tenbaset_phy_pma_config_t *config)
```

Retrieves the current PMA configuration.

Reads the PMA control register and populates the configuration structure.

Parameters

- base – TENBASET_PHY peripheral base address
- config – Pointer to store PMA configuration

```
void TENBASET_PHY_SetPMAConfig(TENBASET_PHY_Type *base, const tenbaset_phy_pma_config_t *config)
```

Configures the PMA operation.

Writes to the PMA control register to set reset, transmission, and loopback modes.

Parameters

- base – TENBASET_PHY peripheral base address
- config – Pointer to PMA configuration

```
void TENBASET_PHY_GetPCSCConfig(TENBASET_PHY_Type *base, tenbaset_phy_pcs_config_t *config)
```

Retrieves the current PCS configuration.

Reads the PCS control register and populates the configuration structure.

Parameters

- base – TENBASET_PHY peripheral base address
- config – Pointer to store PCS configuration

```
void TENBASET_PHY_SetPCSCConfig(TENBASET_PHY_Type *base, const tenbaset_phy_pcs_config_t *config)
```

Configures the PCS operation.

Writes to the PCS control register to set reset and loopback modes.

Parameters

- base – TENBASET_PHY peripheral base address
- config – Pointer to PCS configuration

```
uint16_t TENBASET_PHY_GetRemoteJabberCounter(TENBASET_PHY_Type *base)
```

Gets PCS remote jabber counter.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

Remote jabber counter value.

uint16_t TENBASET_PHY_GetCollisionCounter(TENBASET_PHY_Type *base)

Gets PCS physical collision counter.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

Physical collision counter value.

void TENBASET_PHY_SetWakeConfig(TENBASET_PHY_Type *base, const
tenbaset_phy_wake_config_t *config)

Configures wake settings.

Parameters

- base – TENBASET_PHY peripheral base address.
- config – Pointer to wake configuration structure.

void TENBASET_PHY_TriggerWakeForward(TENBASET_PHY_Type *base)

Triggers a wake-forward pulse.

This function commands the TENBASET_PHY PHY to generate a wake-forward pulse.

Parameters

- base – TENBASET_PHY peripheral base address.

void TENBASET_PHY_TriggerWakeUp(TENBASET_PHY_Type *base)

Triggers WUP sequencer transmission.

Parameters

- base – TENBASET_PHY peripheral base address.

void TENBASET_PHY_AbortWakeUp(TENBASET_PHY_Type *base)

Aborts WUP sequencer when pending.

Parameters

- base – TENBASET_PHY peripheral base address.

void TENBASET_PHY_GetWakeUpStatus(TENBASET_PHY_Type *base,
tenbaset_phy_wup_status_info_t *statusInfo)

Gets WUP status information.

Parameters

- base – TENBASET_PHY peripheral base address.
- statusInfo – Pointer to structure to store WUP status information.

void TENBASET_PHY_EnableInterrupts(TENBASET_PHY_Type *base, uint16_t mask)

Enables TENBASET_PHY interrupts of functional events to the MCU.

Parameters

- base – TENBASET_PHY peripheral base address.
- mask – Interrupt mask (combination of *_tenbaset_phy_interrupt_flags*).

void TENBASET_PHY_DisableInterrupts(TENBASET_PHY_Type *base, uint16_t mask)

Disables TENBASET_PHY interrupts of functional events to the MCU.

Parameters

- base – TENBASET_PHY peripheral base address.
- mask – Interrupt mask (combination of *_tenbaset_phy_interrupt_flags*).

uint16_t TENBASET_PHY_GetInterruptStatus(TENBASET_PHY_Type *base)

Gets interrupt status flags of functional events to the MCU.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

Interrupt status flags.

void TENBASET_PHY_ClearInterruptStatus(TENBASET_PHY_Type *base, uint16_t mask)

Clears interrupt status flags of functional events to the MCU.

Parameters

- base – TENBASET_PHY peripheral base address.
- mask – Interrupt flags to clear.

void TENBASET_PHY_EnableInterrupts2(TENBASET_PHY_Type *base, uint16_t mask)

Enables TENBASET_PHY interrupts of safety related events to the Fault Collection and Control Unit (FCCU).

Parameters

- base – TENBASET_PHY peripheral base address.
- mask – Interrupt mask (combination of _tenbaset_phy_interrupt_flags).

void TENBASET_PHY_DisableInterrupts2(TENBASET_PHY_Type *base, uint16_t mask)

Disables TENBASET_PHY interrupts of safety related events to the Fault Collection and Control Unit (FCCU).

Parameters

- base – TENBASET_PHY peripheral base address.
- mask – Interrupt mask (combination of _tenbaset_phy_interrupt_flags).

uint16_t TENBASET_PHY_GetInterruptStatus2(TENBASET_PHY_Type *base)

Gets interrupt status flags of safety related events to the Fault Collection and Control Unit (FCCU).

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

Interrupt status flags.

void TENBASET_PHY_ClearInterruptStatus2(TENBASET_PHY_Type *base, uint16_t mask)

Clears interrupt status flags of safety related events to the Fault Collection and Control Unit (FCCU).

Parameters

- base – TENBASET_PHY peripheral base address.
- mask – Interrupt flags to clear.

uint32_t TENBASET_PHY_GetPhyId(TENBASET_PHY_Type *base)

Gets PHY identifier.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

OUI part of the 32-bit PHY identifier.

```
void TENBASET_PHY_GetVersionInfo(TENBASET_PHY_Type *base,  
                                tenbaset_phy_version_info_t *versionInfo)
```

Gets version information.

Parameters

- base – TENBASET_PHY peripheral base address.
- versionInfo – Pointer to structure to store version information.

```
uint16_t TENBASET_PHY_GetTransceiverDiagnosticFlags(TENBASET_PHY_Type *base)
```

Gets transceiver diagnostic flags.

Parameters

- base – TENBASET_PHY peripheral base address.

Returns

Diagnostic flags (combination of `_tenbaset_phy_txc_diag_flags`).

```
void TENBASET_PHY_ClearTransceiverDiagnosticFlags(TENBASET_PHY_Type *base, uint16_t  
                                                  diagFlags)
```

Clears transceiver diagnostic flags.

Parameters

- base – TENBASET_PHY peripheral base address.
- diagFlags – Flags to clear (combination of `_tenbaset_phy_txc_diag_flags`).

```
status_t TENBASET_PHY_SetSMIConfig(TENBASET_PHY_Type *base, const  
                                   tenbaset_phy_smi_config_t *config)
```

Configures the SMI (Serial Management Interface) parameters.

This function sets up the SMI interface configuration, including:

- MDIO hold time
- Preamble generation
- MDC clock frequency

Parameters

- base – TENBASET_PHY peripheral base address
- config – Pointer to SMI configuration structure

Returns

`status_t` Operation status

- `kStatus_Success`: Configuration successful
- `kStatus_InvalidArgument`: Invalid speed parameter

```
bool TENBASET_PHY_GetSMIStatus(TENBASET_PHY_Type *base,  
                               tenbaset_phy_smi_error_code_t *errorCode)
```

Retrieves the current SMI status.

Checks if the SMI is ready and optionally retrieves the current error code.

Parameters

- base – TENBASET_PHY peripheral base address
- errorCode – Pointer to store error code (can be NULL if error code not needed)

Returns

`bool`

- true: SMI is ready
- false: SMI is not ready

`status_t` TENBASET_PHY_SMIRead(TENBASET_PHY_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *data)

Reads a register from an external PMD transceiver via SMI.

Performs an SMI read operation with timeout handling.

Parameters

- base – TENBASET_PHY peripheral base address
- phyAddr – PHY address of PMD transceiver
- regAddr – Register address to read
- data – Pointer to store the read register value

Returns

`status_t` Operation status

- kStatus_Success: Read successful
- kStatus_Timeout: SMI not ready within timeout period

`status_t` TENBASET_PHY_SMIWrite(TENBASET_PHY_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

Writes a register to an external PMD transceiver via SMI.

Performs an SMI write operation with timeout handling.

Parameters

- base – TENBASET_PHY peripheral base address
- phyAddr – PHY address of PMD transceiver
- regAddr – Register address to write
- data – Value to write to the register

Returns

`status_t` Operation status

- kStatus_Success: Write successful
- kStatus_Timeout: SMI not ready within timeout period

`struct __tenbaset_phy_plca_config`

#include <fsl_tenbaset_phy.h> TENBASET_PHY PLCA configuration structure.

Public Members

`uint8_t` nodeId

PLCA node identifier

`uint8_t` nodeCount

Total number of nodes in network

`uint8_t` toTimer

Transmit opportunity timer value

`uint8_t` burstTimer

Burst timer value

uint8_t maxBurstCount

Maximum burst count

bool enable

Enable PLCA

struct __tenbaset_phy_pma_config

#include <fsl_tenbaset_phy.h> PMA configuration structure.

Public Members

bool resetEnable

Reset the PMA and PCS layers

bool transmissionEnable

Enable transmission

bool loopbackEnable

Enable PMA loopback mode

struct __tenbaset_phy_pcs_config

#include <fsl_tenbaset_phy.h> PCS configuration structure.

Public Members

bool resetEnable

Reset the PCS layer

bool loopbackEnable

Enable PCS loopback mode

struct __tenbaset_phy_wake_config

#include <fsl_tenbaset_phy.h> TENBASET_PHY wake configuration structure.

Public Members

bool suspendWakeEnable

Enable wake-up to WAITINIT mode by SUSPEND detection

bool remoteWakeEnable

Enable wake-up to WAITINIT mode by remote-wake-event

bool localWakeEnable

Enable wake-up to WAITINIT mode by local-wake-event

bool remoteWakeForwardEnable

Enable wake forwarding of a remote-wake-event

bool localWakeWupEnable

Enable WUP transmission by local-wake-event

uint8_t wupTimeout

WUP timeout value (WUP time-out = (WUPTIMEOUT + 1) * 10ms)

struct __tenbaset_phy_wup_status_info

#include <fsl_tenbaset_phy.h> TENBASET_PHY WUP status structure.

Public Members

tenbaset_phy_wup_status_t status

WUP sequencer status

tenbaset_phy_wup_error_code_t errorCode

Error code of previous WUP sequence

struct *_tenbaset_phy_version_info*

#include <fsl_tenbaset_phy.h> TENBASET_PHY version information structure.

Public Members

uint8_t majorVersion

Major version number

uint8_t minorVersion

Minor version number

uint8_t customerVersion

Customer version number

struct *_tenbaset_phy_smi_config*

#include <fsl_tenbaset_phy.h> SMI configuration structure.

Public Members

tenbaset_phy_smi_hold_time_t holdTime

MDIO hold time

bool preambleEnable

Enable SMI preamble generation

uint8_t speed

MDC clock frequency divisor (0-63)

struct *_tenbaset_phy_config*

#include <fsl_tenbaset_phy.h> TENBASET_PHY configuration structure.

Public Members

tenbaset_phy_plca_config_t plcaConfig

PLCA configuration

tenbaset_phy_wake_config_t wakeConfig

Wake configuration

uint16_t interruptMask

Mask of enabled interrupts

struct *_tenbaset_phy_handle*

#include <fsl_tenbaset_phy.h> TENBASET_PHY handle structure.

Public Members

tenbaset_phy_callback_t callback

Callback function

`void *userData`
User data for callback

`bool initialized`
Initialization status

2.69 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 domians whose NSE bits to clear.

```
void TRDC__MrcSetRegionDescriptorConfig(TRDC_Type *base, const  
                                       trdc_mrc_region_descriptor_config_t *config)
```

Sets the configuration for one of the region descriptor per domain per MRC instnce.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.

- config – Pointer to region descriptor configuration structure.

static inline void TRDC_SetMbcGlobalValid(TRDC_Type *base)

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

void TRDC_GetMbcHardwareConfig(TRDC_Type *base, *trdc_slave_memory_hardware_config_t* *config, uint8_t mbcIdx, uint8_t slvIdx)

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

void TRDC_MbcSetNseUpdateConfig(TRDC_Type *base, const *trdc_mbc_nse_update_config_t* *config, uint8_t mbcIdx)

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be updateby NSE set/clear.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

void TRDC_MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

void TRDC_MbcWordNseClear(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

void TRDC_MbcNseClearAll(TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slave)

Clears all configuration words' NSE bits of the selected domain and memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

```
void TRDC_MbcSetMemoryAccessConfig(TRDC_Type *base, const
                                   trdc_memory_access_control_config_t *config, uint8_t
                                   mbcIdx, uint8_t rgdIdx)
```

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

```
void TRDC_MbcSetMemoryBlockConfig(TRDC_Type *base, const
                                  trdc_mbc_memory_block_config_t *config)
```

Sets the configuration for one of the memory block per domain per MBC instnce.

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.

2.70 Trdc_core

typedef struct *_TRDC_General_Type* TRDC_General_Type

TRDC general configuration register definition.

typedef struct *_TRDC_FLW_Type* TRDC_FLW_Type

TRDC flash logical control register definition.

typedef struct *_TRDC_DomainError_Type* TRDC_DomainError_Type

TRDC domain error register definition.

typedef struct *_TRDC_DomainAssignment_Type* TRDC_DomainAssignment_Type

TRDC master domain assignment register definition.

typedef struct *_TRDC_MBC_Type* TRDC_MBC_Type

TRDC MBC control register definition.

typedef struct *_TRDC_MRC_Type* TRDC_MRC_Type

TRDC MRC control register definition. MRC_DOM0_RGD_W[region][word].

TRDC_GENERAL_BASE(base)

TRDC base address convert macro.

TRDC_FLW_BASE(base)

TRDC_DOMAIN_ERROR_BASE(base)

TRDC_DOMAIN_ASSIGNMENT_BASE(base)

TRDC_MBC_BASE(base, instance)

TRDC_MRC_BASE(base, instance)

struct *_TRDC_General_Type*

#include <fsl_trdc_core.h> TRDC general configuration register definition.

Public Members

__IO uint32_t TRDC_CR

TRDC Register, offset: 0x0

__I uint32_t TRDC_HWCFG0

TRDC Hardware Configuration Register 0, offset: 0xF0


```

__IO uint32_t TRDC_HWCFG1
    TRDC Hardware Configuration Register 1, offset: 0xF4
__IO uint32_t TRDC_HWCFG2
    TRDC Hardware Configuration Register 2, offset: 0xF8
__IO uint32_t TRDC_HWCFG3
    TRDC Hardware Configuration Register 3, offset: 0xFC
__IO uint8_t DACFG [8]
    Domain Assignment Configuration Register, array offset: 0x100, array step: 0x1
__IO uint32_t TRDC_IDAU_CR
    TRDC IDAU Control Register, offset: 0x1C0

```

```
struct __TRDC_FLW_Type
```

```
#include <fsl_trdc_core.h> TRDC flash logical control register definition.
```

Public Members

```

__IO uint32_t TRDC_FLW_CTL
    TRDC FLW Control, offset: 0x1E0
__IO uint32_t TRDC_FLW_PBASE
    TRDC FLW Physical Base, offset: 0x1E4
__IO uint32_t TRDC_FLW_ABASE
    TRDC FLW Array Base, offset: 0x1E8
__IO uint32_t TRDC_FLW_BCNT
    TRDC FLW Block Count, offset: 0x1EC

```

```
struct __TRDC_DomainError_Type
```

```
#include <fsl_trdc_core.h> TRDC domain error register definition.
```

Public Members

```

__IO uint32_t TRDC_FDIS
    TRDC Fault Domain ID, offset: 0x1FC
__IO uint32_t TRDC_DERRLOC [16]
    TRDC Domain Error Location Register, array offset: 0x200, array step: 0x4

```

```
struct __TRDC_DomainAssignment_Type
```

```
#include <fsl_trdc_core.h> TRDC master domain assignment register definition.
```

Public Members

```

__IO uint32_t PID [8]
    Process Identifier, array offset: 0x700, array step: 0x4

```

```
struct __TRDC_MBC_Type
```

```
#include <fsl_trdc_core.h> TRDC MBC control register definition.
```

Public Members

- ___I uint32_t MBC_MEM_GLBCFG [4]
MBC Global Configuration Register, array offset: 0x10000, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_NSE_BLK_INDEX
MBC NonSecure Enable Block Index, array offset: 0x10010, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_SET
MBC NonSecure Enable Block Set, array offset: 0x10014, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_CLR
MBC NonSecure Enable Block Clear, array offset: 0x10018, array step: 0x2000
- ___O uint32_t MBC_NSE_BLK_CLR_ALL
MBC NonSecure Enable Block Clear All, array offset: 0x1001C, array step: 0x2000
- ___IO uint32_t MBC_MEMN_GLBAC [8]
MBC Global Access Control, array offset: 0x10020, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x101D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM0_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x101F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10340, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM1_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10380, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x103D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM1_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x103F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x105D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM2_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x105F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10640, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM3_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10740, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10780, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x107D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM3_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x107F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10840, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10940, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10980, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x109D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM4_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x109F0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM5_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10A40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM5_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10D80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM6_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10DD0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM6_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10DF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x10E40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x10F40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10F80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x10FD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM7_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x10FF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11040, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11140, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11180, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111C8, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM8_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x111D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM8_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x111F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11240, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11340, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11380, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x113D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM9_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x113F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11440, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11540, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11580, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115A0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM10_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115A8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM10_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115C8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM10_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x115D0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM10_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x115F0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11640, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11740, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11780, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117A0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117A8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117C8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x117D0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM11_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x117F0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM12_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11840, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM12_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11940, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM12_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11980, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM12_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119A0, array step: index*0x2000, index2*0x4

- ___IO uint32_t MBC_DOM12_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119A8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119C8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x119D0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM12_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x119F0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11A40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11B40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11B80, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM1_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BA0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BA8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM2_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BC8, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11BD0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM13_MEM3_BLK_NSE_W [2]
MBC Memory Block NonSecure Enable Word, array offset: 0x11BF0, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_CFG_W [64]
MBC Memory Block Configuration Word, array offset: 0x11C40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM0_BLK_NSE_W [16]
MBC Memory Block NonSecure Enable Word, array offset: 0x11D40, array step: index*0x2000, index2*0x4
- ___IO uint32_t MBC_DOM14_MEM1_BLK_CFG_W [8]
MBC Memory Block Configuration Word, array offset: 0x11D80, array step: index*0x2000, index2*0x4

```
__IO uint32_t MBC_DOM14_MEM1_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11DA0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM2_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11DA8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM2_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11DC8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM3_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11DD0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM14_MEM3_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11DF0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM0_BLK_CFG_W [64]
    MBC Memory Block Configuration Word, array offset: 0x11E40, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM0_BLK_NSE_W [16]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11F40, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM1_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11F80, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM1_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FA0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM2_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11FA8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM2_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FC8, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_CFG_W [8]
    MBC Memory Block Configuration Word, array offset: 0x11FD0, array step: index*0x2000, index2*0x4

__IO uint32_t MBC_DOM15_MEM3_BLK_NSE_W [2]
    MBC Memory Block NonSecure Enable Word, array offset: 0x11FF0, array step: index*0x2000, index2*0x4
```

```
struct _TRDC_MRC_Type
```

```
#include <fsl_trdc_core.h> TRDC MRC control register definition.
MRC_DOM0_RGD_W[region][word].
```

Public Members

```
__IO uint32_t MRC_GLBCFG
    MRC Global Configuration Register, array offset: 0x14000, array step: 0x1000
```

```

__IO uint32_t MRC_NSE_RGN_INDIRECT
    MRC NonSecure Enable Region Indirect, array offset: 0x14010, array step: 0x1000
__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 __unnamed73__

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

2.71 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

2.72 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.

2.73 UTICK: MictoTick Timer Driver

`void UTICK_Init(UTICK_Type *base)`

Initializes an UTICK by turning its bus clock on.

`void UTICK_Deinit(UTICK_Type *base)`

Deinitializes a UTICK instance.

This function shuts down Utick bus clock

Parameters

- `base` – UTICK peripheral base address.

`uint32_t UTICK_GetStatusFlags(UTICK_Type *base)`

Get Status Flags.

This returns the status flag

Parameters

- `base` – UTICK peripheral base address.

Returns

status register value

`void UTICK_ClearStatusFlags(UTICK_Type *base)`

Clear Status Interrupt Flags.

This clears intr status flag

Parameters

- base – UTICK peripheral base address.

Returns

none

```
void UTICK_SetTick(UTICK_Type *base, utick_mode_t mode, uint32_t count, utick_callback_t cb)
```

Starts UTICK.

This function starts a repeat/onetime countdown with an optional callback

Parameters

- base – UTICK peripheral base address.
- mode – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
- count – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
- cb – UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

Returns

none

```
void UTICK_HandleIRQ(UTICK_Type *base, utick_callback_t cb)
```

UTICK Interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in UTICK_SetTick()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – UTICK peripheral base address.
- cb – callback scheduled for this instance of UTICK

Returns

none

```
FSL_UTICK_DRIVER_VERSION
```

UTICK driver version 2.0.5.

```
enum _utick_mode
```

UTICK timer operational mode.

Values:

```
enumerator kUTICK_Onetime
```

Trigger once

```
enumerator kUTICK_Repeat
```

Trigger repeatedly

```
typedef enum _utick_mode utick_mode_t
```

UTICK timer operational mode.

```
typedef void (*utick_callback_t)(void)
```

UTICK callback function.

2.74 VREF: Voltage Reference Driver

void VREF_Init(VREF_Type *base, const *vref_config_t* *config)

Enables the clock gate and configures the VREF module according to the configuration structure.

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up *vref_config_t* parameters and how to call the VREF_Init function by passing in these parameters.

```
vref_config_t vrefConfig;
VREF_GetDefaultConfig(&vrefConfig);
vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
VREF_Init(VREF, &vrefConfig);
```

Parameters

- *base* – VREF peripheral address.
- *config* – Pointer to the configuration structure.

void VREF_Deinit(VREF_Type *base)

Stops and disables the clock for the VREF module.

This function should be called to shut down the module. This is an example.

```
vref_config_t vrefUserConfig;
VREF_GetDefaultConfig(&vrefUserConfig);
VREF_Init(VREF, &vrefUserConfig);
...
VREF_Deinit(VREF);
```

Parameters

- *base* – VREF peripheral address.

void VREF_GetDefaultConfig(*vref_config_t* *config)

Initializes the VREF configuration structure.

This function initializes the VREF configuration structure to default values. This is an example.

```
config->bufferMode = kVREF_ModeHighPowerBuffer;
config->enableInternalVoltageRegulator = true;
config->enableChopOscillator          = true;
config->enableHCBandgap                = true;
config->enableCurvatureCompensation    = true;
config->enableLowPowerBuff             = true;
```

Parameters

- *config* – Pointer to the initialization structure.

void VREF_SetVrefTrimVal(VREF_Type *base, uint8_t trimValue)

Sets a TRIM value for the accurate 1.0V bandgap output.

This function sets a TRIM value for the reference voltage. It will trim the accurate 1.0V bandgap by 0.5mV each step.

Parameters

- *base* – VREF peripheral address.
- *trimValue* – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

```
void VREF_SetTrim21Val(VREF_Type *base, uint8_t trim21Value)
```

Sets a TRIM value for the accurate buffered VREF output.

This function sets a TRIM value for the reference voltage. If buffer mode be set to other values (Buf21 enabled), it will trim the VREF_OUT by 0.1V each step from 1.0V to 2.1V.

Note: When Buf21 is enabled, the value of UTRIM[TRIM2V1] should be ranged from 0b0000 to 0b1011 in order to trim the output voltage from 1.0V to 2.1V, other values will make the VREF_OUT to default value, 1.0V.

Parameters

- base – VREF peripheral address.
- trim21Value – Value of the trim register to set the output reference voltage (maximum 0xF (4-bit)).

```
uint8_t VREF_GetVrefTrimVal(VREF_Type *base)
```

Reads the trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[VREFTRIM] (13:8)

Parameters

- base – VREF peripheral address.

Returns

6-bit value of trim setting.

```
uint8_t VREF_GetTrim21Val(VREF_Type *base)
```

Reads the VREF 2.1V trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[TRIM2V1] (3:0),

Parameters

- base – VREF peripheral address.

Returns

4-bit value of trim setting.

```
static inline void VREF_SetAdcNegChannelRef(VREF_Type *base, bool enable)
```

Decides whether to enable the ADC negative channel to receive the reference voltage.

Parameters

- base – VREF peripheral address.
- enable – Enable/disable ADC negative channel reference voltage.
 - **true** Enable ADC negative channel reference voltage.
 - **false** Disable ADC negative channel reference voltage.

```
static inline void VREF_SetAdcPosChannelRef(VREF_Type *base, bool enable)
```

Decides whether to enable the ADC positive channel to receive the reference voltage.

Parameters

- base – VREF peripheral address.
- enable – Enable/disable ADC positive channel reference voltage.
 - **true** Enable ADC positive channel reference voltage.
 - **false** Disable ADC positive channel reference voltage.

FSL_VREF_DRIVER_VERSION

Version 2.5.0.

enum *_vref_buffer_mode*

VREF buffer modes.

Values:

enumerator *kVREF_ModeBandgapOnly*

Bandgap enabled/standby.

enumerator *kVREF_ModeLowPowerBuffer*

Low-power buffer mode enabled

enumerator *kVREF_ModeHighPowerBuffer*

High-power buffer mode enabled

typedef enum *_vref_buffer_mode* *vref_buffer_mode_t*

VREF buffer modes.

typedef struct *_vref_config* *vref_config_t*

The description structure for the VREF module.

struct *_vref_config*

#include <fsl_vref.h> The description structure for the VREF module.

Public Members

vref_buffer_mode_t *bufferMode*

Buffer mode selection

bool *enableInternalVoltageRegulator*

Provide additional supply noise rejection.

bool *enableChopOscillator*

Enable Chop oscillator.

bool *enableHCBandgap*

Enable High-Accurate bandgap.

bool *enableCurvatureCompensation*

Enable second order curvature compensation.

bool *enableLowPowerBuff*

Provides bias current for other peripherals.

2.75 WAKETIMER: WAKETIMER Driver

void *WAKETIMER_Init*(*WAKETIMER_Type* *base, const *waketimer_config_t* *config)

Initializes an WAKETIMER.

This function initializes the WAKETIMER.

Parameters

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

void WAKETIMER_Deinit(WAKETIMER_Type *base)

Deinitializes a WAKETIMER instance.

This function deinitialize the WAKETIMER.

Parameters

- base – WAKETIMER peripheral base address.

void WAKETIMER_GetDefaultConfig(*waketimer_config_t* *config)

Fills in the WAKETIMER configuration structure with the default settings.

The default values are:

```
config->enableInterrupt = true;
config->enableOSCDivide = true;
config->callback        = NULL;
```

Parameters

- config – Pointer to the user configuration structure.

void WAKETIMER_EnableInterrupts(WAKETIMER_Type *base, uint32_t mask)

Enables the selected WAKETIMER interrupts.

Parameters

- base – WAKETIMER peripheral base address
- mask – Mask value for interrupt events. See to `_waketimer_interrupt_enable`

void WAKETIMER_DisableInterrupts(WAKETIMER_Type *base, uint32_t mask)

Enables the selected WAKETIMER interrupts.

Parameters

- base – WAKETIMER peripheral base address
- mask – Mask value for interrupt events. See to `_waketimer_interrupt_enable`

void WAKETIMER_ClearStatusFlags(WAKETIMER_Type *base, uint32_t mask)

Clear Status Interrupt Flag.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

- base – WAKETIMER peripheral base address.
- mask – Mask value for flags to be cleared. See to `_waketimer_status_flags`.

Returns

none

void WAKETIMER_SetCallback(WAKETIMER_Type *base, *waketimer_callback_t* callback)

Receive noticification when waketime countdown.

If the interrupt for the waketime countdown is enabled, then a callback can be registered which will be invoked when the event is triggered

Parameters

- base – WAKETIMER peripheral base address
- callback – Function to invoke when the event is triggered

static inline void WAKETIMER_HaltTimer(WAKETIMER_Type *base)

Halt and clear timer counter.

This halt and clear timer counter.

Parameters

- base – WAKETIMER peripheral base address.

Returns

none

static inline void WAKETIMER_StartTimer(WAKETIMER_Type *base, uint32_t value)

Set timer counter.

This set the timer counter and start the timer countdown.

Parameters

- base – WAKETIMER peripheral base address.
- value – countdown value.

Returns

none

uint32_t WAKETIMER_GetCurrentTimerValue(WAKETIMER_Type *base)

Get current timer count value from WAKETIMER.

This function will get a decimal timer count value. The RAW value of timer count is gray code format, will be translated to decimal data internally.

Parameters

- base – WAKETIMER peripheral base address.

Returns

Value of WAKETIMER which will be formatted to decimal value.

FSL_WAKETIMER_DRIVER_VERSION

WAKETIMER driver version.

enum _waketimer_status_flags

WAKETIMER status flags.

Values:

enumerator kWAKETIMER_WakeFlag

Wake Timer Status Flag, sets wake timer has timed out.

enum _waketimer_interrupt_enable

Define interrupt switchers of the module.

Values:

enumerator kWAKETIMER_WakeInterruptEnable

Generate interrupt requests when WAKE_FLAG is asserted.

typedef void (*waketimer_callback_t)(void)

waketimer callback function.

typedef struct _waketimer_config waketimer_config_t

WAKETIMER configuration structure.

This structure holds the configuration settings for the WAKETIMER peripheral. To initialize this structure to reasonable defaults, call the WAKETIMER_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

struct __waketimer_config

#include <fsl_waketimer.h> WAKETIMER configuration structure.

This structure holds the configuration settings for the WAKETIMER peripheral. To initialize this structure to reasonable defaults, call the WAKETIMER_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

bool enableOSCDivide

true: Enable OSC Divide. false: Disable OSC Divide.

bool enableInterrupt

true: Enable interrupt. false: Disable interrupt.

waketimer_callback_t callback

timer countdown callback.

2.76 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 config pins options, including edge detection wakeup event and operate mode.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin. See Reference Manual for the details.
- config – Pointer to wuu_external_wakeup_pin_config_t structure.

void WUU_ClearExternalWakeupPinsConfig(WUU_Type *base, uint8_t pinIndex)

Disable and clear external wakeup pin settings.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin.

static inline uint32_t WUU_GetExternalWakeUpPinsFlag(WUU_Type *base)

Gets External Wakeup pin flags.

This function return the external wakeup pin flags.

Parameters

- base – WUU peripheral base address.

Returns

Wakeup flags for all external wakeup pins.

static inline void WUU_ClearExternalWakeUpPinsFlag(WUU_Type *base, uint32_t mask)

Clears External WakeUp Pin flags.

This function clears external wakeup pins flags based on the mask.

Parameters

- base – WUU peripheral base address.
- mask – The mask of Wakeup pin index to be cleared.

```
void WUU_SetInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                         wuu_internal_wakeup_module_event_t event)
```

Config Internal modules' event as the wake up sources.

This function configs the internal modules event as the wake up sources.

Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – Select interrupt or DMA/Trigger of the internal module as the wake up source.

```
void WUU_ClearInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex,  
                                           wuu_internal_wakeup_module_event_t event)
```

Disable an on-chip internal modules' event as the wakeup sources.

Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – The event(interrupt or DMA/trigger) of the internal module to disable.

```
void WUU_SetPinFilterConfig(WUU_Type *base, uint8_t filterIndex, const  
                           wuu_pin_filter_config_t *config)
```

Configs and Enables Pin filters.

This function configs Pin filter, including pin select, filter operate mode filter wakeup event and filter edge detection.

Parameters

- base – WUU peripheral base address.
- filterIndex – The index of the pin filter.
- config – Pointer to *wuu_pin_filter_config_t* structure.

```
bool WUU_GetPinFilterFlag(WUU_Type *base, uint8_t filterIndex)
```

Gets the pin filter configuration.

This function gets the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

```
void WUU_ClearPinFilterFlag(WUU_Type *base, uint8_t filterIndex)
```

Clears the pin filter configuration.

This function clears the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

bool WUU_GetExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0. return True if the specific pin is a wakeup source.

void WUU_ClearExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0.

FSL_WUU_DRIVER_VERSION

Defines WUU driver version 2.4.2.

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 _wuui_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 _wuui_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 _wuui_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 _wuui_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.

wuu_filter_edge_t edge
    The edge of the pin digital filter.

wuu_filter_event_t event
    The event of the filter output.

wuu_filter_mode_t mode
    The mode of the filter operate.
```

2.77 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 $TWDCLK \cdot 256 \cdot 4$. If `enableWatchdogProtect` flag is true in `wwdt_config_t` config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the `WDTOF` flag.

Parameters

- `base` – WWDT peripheral base address
- `timeoutCount` – WWDT timeout value, count of WWDT clock tick.

```
static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)
```

Sets the WWDT window value.

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set `windowValue` to `0xFFFFF` (maximum possible timer value) so windowing is not in effect.

Parameters

- `base` – WWDT peripheral base address
- `windowValue` – WWDT window value.

```
void WWDT_Refresh(WWDT_Type *base)
```

Refreshes the WWDT timer.

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

- `base` – WWDT peripheral base address

```
FSL_WWDT_DRIVER_VERSION
```

Defines WWDT driver version.

```
WWDT_FIRST_WORD_OF_REFRESH
```

First word of refresh sequence

```
WWDT_SECOND_WORD_OF_REFRESH
```

Second word of refresh sequence

```
enum _wwdt_status_flags_t
```

WWDT status flags.

This structure contains the WWDT status flags for use in the WWDT functions.

Values:

```
enumerator kWWDT_TimeoutFlag
```

Time-out flag, set when the timer times out

```
enumerator kWWDT_WarningFlag
```

Warning interrupt flag, set when timer is below the value `WDWARNINT`

```
typedef struct _wwdt_config wwdt_config_t
```

Describes WWDT configuration structure.

```
struct _wwdt_config
```

#include <fsl_wwdt.h> Describes WWDT configuration structure.

Public Members

`bool enableWwdt`

Enables or disables WWDT

`bool enableWatchdogReset`

true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset

`bool enableWatchdogProtect`

true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time

`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:

$$N * T_{char}$$

where:

- *N* is equal to the length of the receive FIFO queue (configured by the `FMSTR_COMM_QUEUE_SIZE` macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);  
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see [Driver interrupt modes](#)), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the `FMSTR_USE_RECORDER` configuration option (minus one). An exception to this requirement is the recorder of instance 0 which may be automatically configured by `FMSTR_Init` when the *freemaster_cfg.h* configuration file defines the `FMSTR_REC_BUFF_SIZE` and `FMSTR_REC_TIMEBASE` options.

For more information, see [Configurable items](#).

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the *FMSTR_USE_TSA* macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-language symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type) /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type) /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type) /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT n	Unsigned integer type of size n bits ($n=8,16,32,64$)
FMSTR_TSA_SINT n	Signed integer type of size n bits ($n=8,16,32,64$)
FMSTR_TSA_FRAC n	Fractional number of size n bits ($n=16,32,64$).
FMSTR_TSA_FRAC_Q(m,n)	Signed fractional number in general Q form ($m+n+1$ total bits)
FMSTR_TSA_FRAC_UQ(m,n)	Unsigned fractional number in general UQ form ($m+n$ total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FM-STR_TSA_USERTYPE($name$)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files") /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index)) /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")
```

(continues on next page)

(continued from previous page)

```
/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR ↵
↵ tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object

- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
 - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
 - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
 - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the *FMSTR_USE_TSA_DYNAMIC* configuration option and when the *FMSTR_SetUpTsaBuff* function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See [TSA table definition](#) for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the *FMSTR_APPCMDRESULT_NOCMD* constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the *FMSTR_AppCmdAck* call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The *FMSTR_GetAppCmd* function does not report the commands for which a callback handler function exists. If the *FMSTR_GetAppCmd* function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns *FMSTR_APPCMDRESULT_NOCMD*.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see [FMSTR_GetAppCmd](#)).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↳ PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↳
FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_XXX and FMSTR_PIPE_SIZE_XXX constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the *nGranularity* value equal to the *nLength* value, all data are considered as one chunk which is either written successfully as a whole or not at all. The *nGranularity* value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The `readGranularity` argument can be used to copy the data in larger chunks in the same way as described in the `FMSTR_PipeWrite` function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
<i>FM-STR_ADDR</i> For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
<i>FM-STR_SIZE</i> It is required that this type is unsigned and at least 16 bits wide integer.	Data type used to hold the memory block size.
<i>FM-STR_BOOL</i> This type is used only in zero/non-zero conditions in the driver code.	Data type used as a general boolean type.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command code.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to create the Application Command data buffer.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command result code.

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

<i>FM-STR_TSA_TII</i>	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables. By default, this is defined as <i>FM-STR_SIZE</i> .
<i>FM-STR_TSA_TS</i>	Data type used to hold a memory block size, as used in the TSA descriptors. By default, this is defined as <i>FM-STR_SIZE</i> .

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

<i>FM-STR_HPIPE</i>	Pipe handle that identifies the open-pipe object. Generally, this is a pointer to a void type.
<i>FM-STR_PIPE_PC</i>	Integer type required to hold at least 7 bits of data. Generally, this is an unsigned 8-bit or 16-bit type.
<i>FM-STR_PIPE_SI</i>	Integer type required to hold at least 16 bits of data. This is used to store the data buffer sizes.
<i>FM-STR_PPIPEF</i>	Pointer to the pipe handler function. See FM-STR_PipeOpen for more details.

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

<i>FMSTR_U8</i>	The smallest memory entity.
On the vast majority of platforms, this is an unsigned 8-bit integer.	
On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.	
<i>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