MCUXpresso SDK Documentation

Release 25.12.00

NXP
Dec 18, 2025

Table of contents

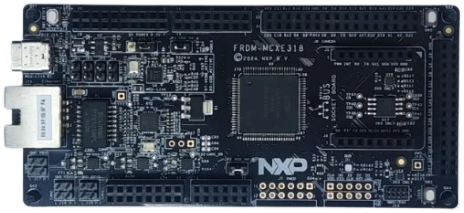
This documentation contains information specific to the frdmmcxe31b board.

Chapter 1

FRDM-MCXE31B

1.1 Overview

The FRDM-MCXE31B board is a design and evaluation platform based on the NXP MCXE31B microcontroller (MCU). NXP MCXE31B MCU based on an Arm Cortex- M7 core, running at speeds of up to 160 MHz with a 2.97–5.5 V supply. The FRDM-MCXE31B board consists of one MCXE31B device with a 64 Mbit external serial flash (provided by Winbond). The board also features FXLS8974CFR3 I2C accelerometer sensor, one NMH1000 I2C Magnetic switch, three TJA1057GTK/3Z CAN PHY, Ethernet PHY, RGB LED, push buttons, and MCU-Link debug probe circuit. The board is compatible with the Arduino shield modules, Pmod boards, and mikroBUS. For debugging the MCXE31B MCU, the FRDM-MCXE31B board uses an onboard (OB) debug probe, MCU-Link OB, which is based on another NXP MCU: LPC55S16.



MCU device and part on board is shown below:

- Device: MCXE31B
- PartNumber: MCXE31BMPB

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.

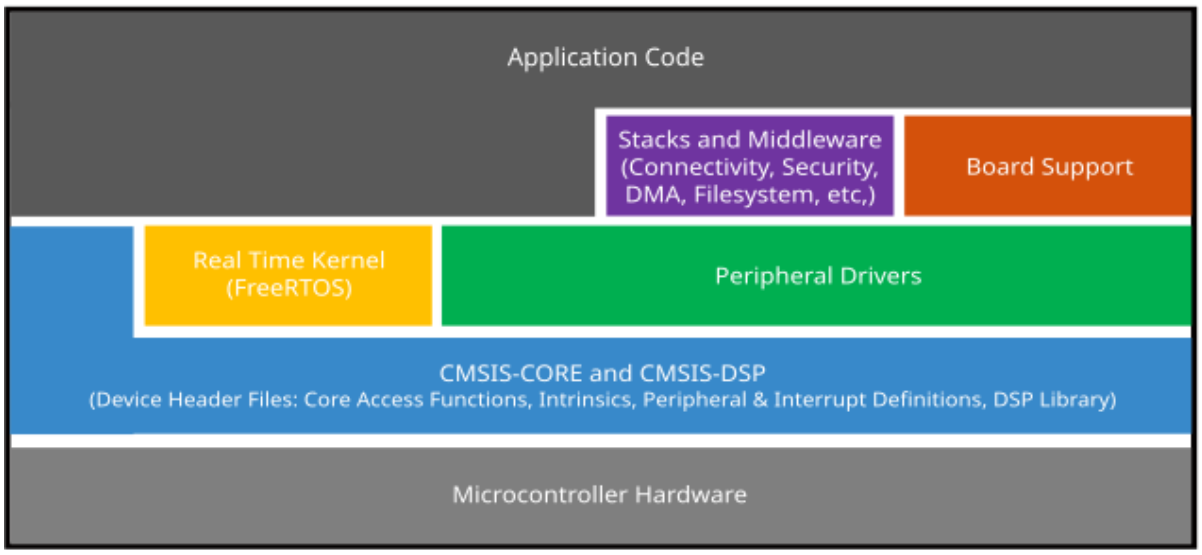
Note: The Repository-Layout SDK package was first introduced in version 25.09.00, but initially only for MCXW23x platforms.

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

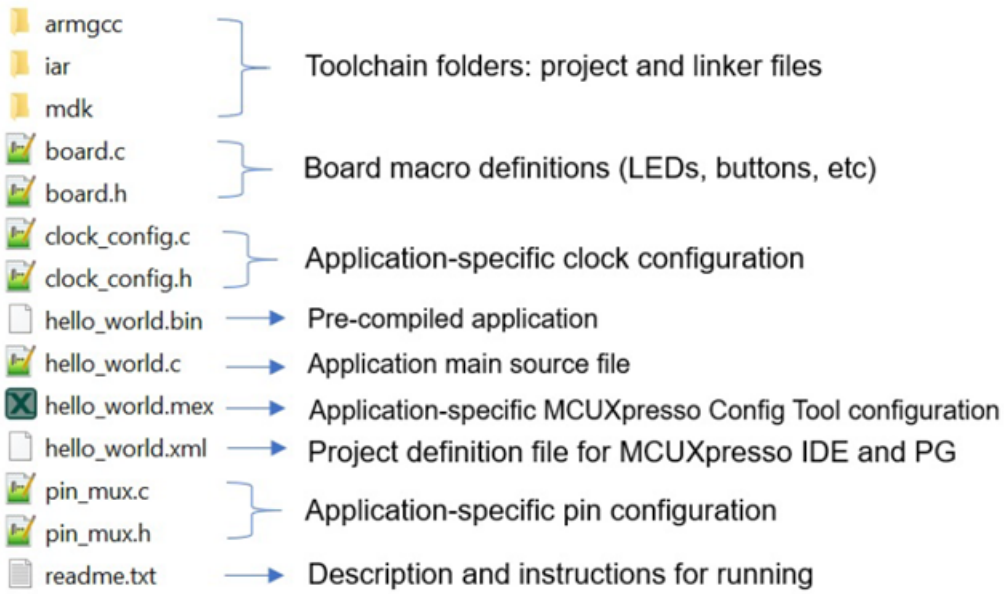
but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).

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

For examples containing middleware/stacks or an RTOS, there are references to the appropriate source code. Middleware source files are located in the middleware folder and RTOSes are in the rtos folder. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

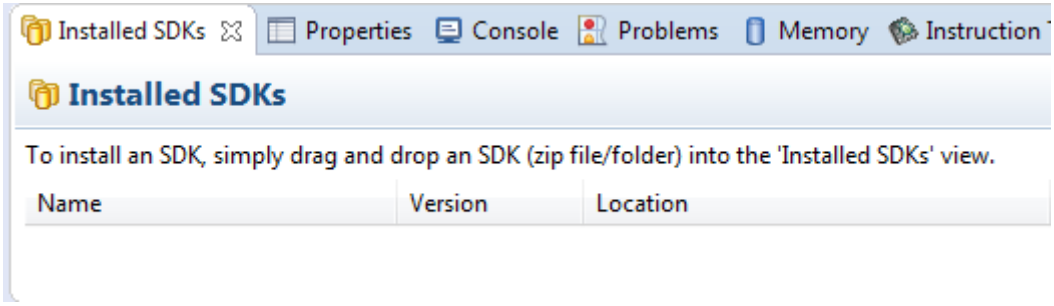
Run a demo using MCUXpresso IDE **Note:** Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

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.

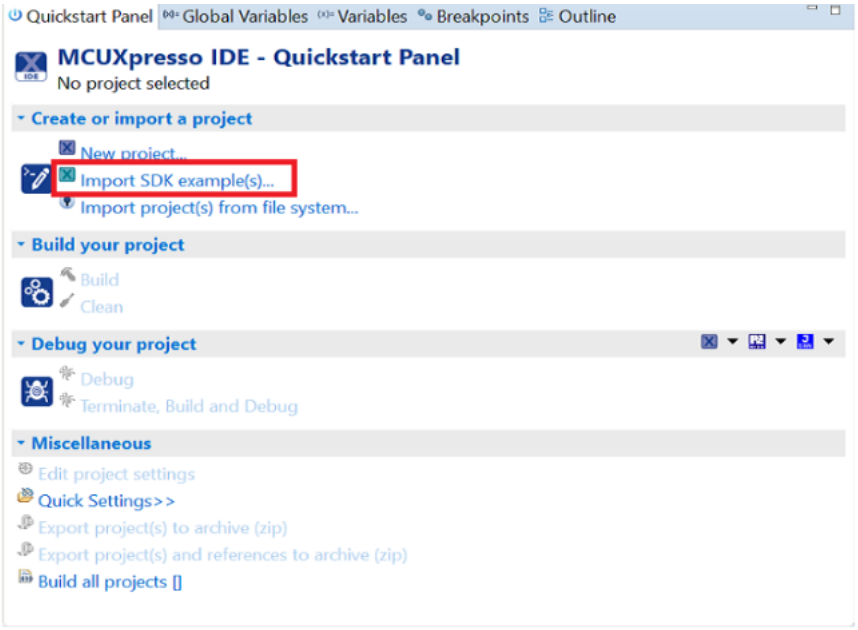
Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace.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

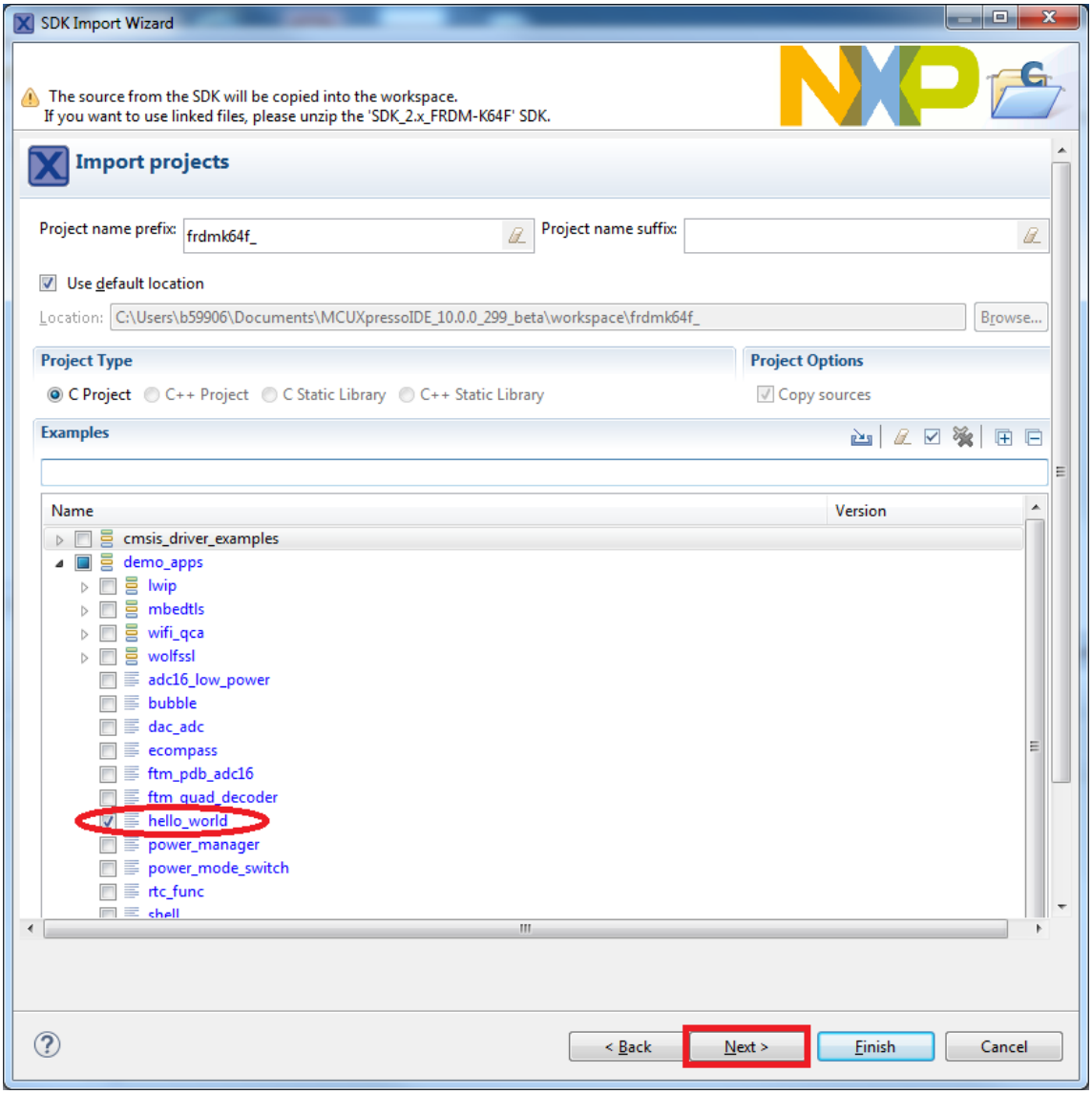
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



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



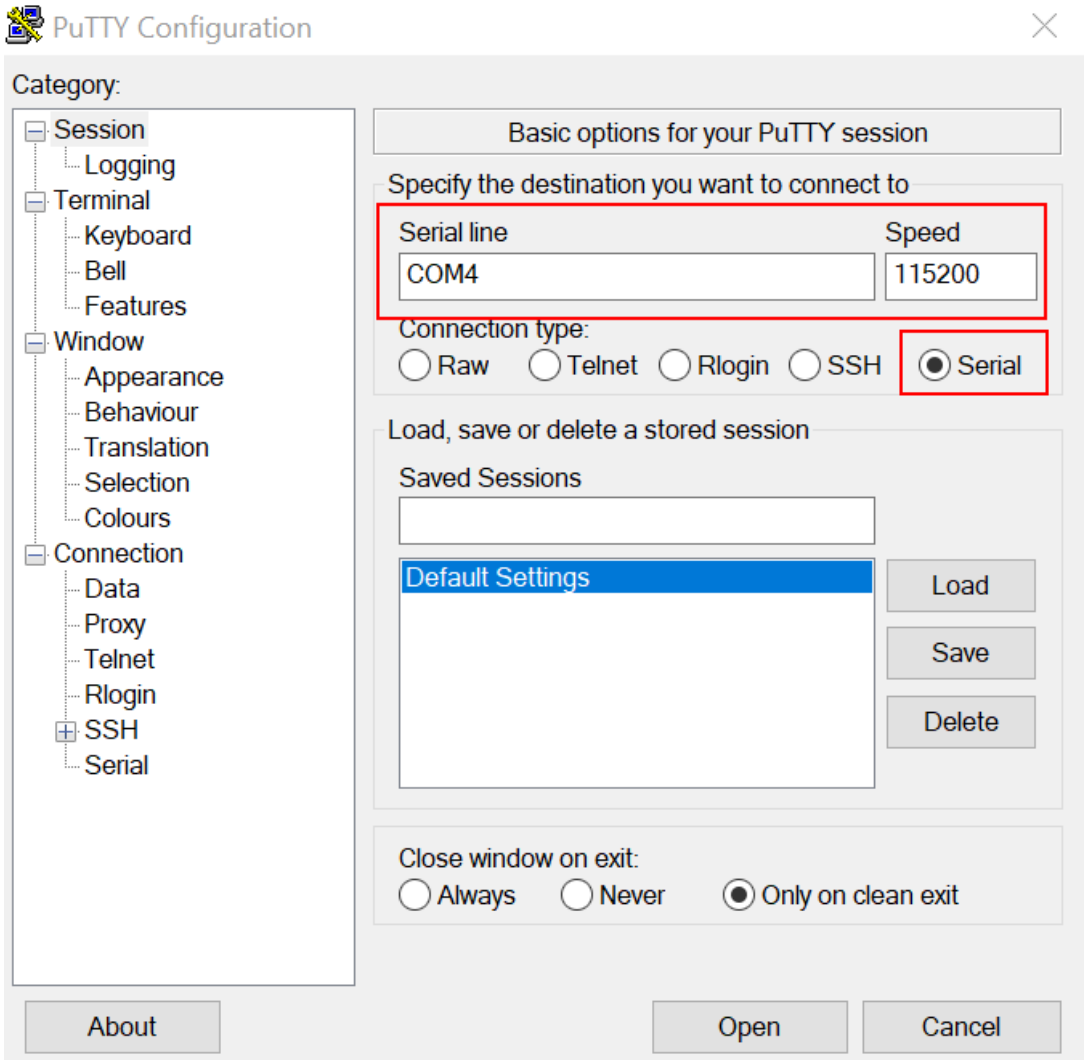
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminalfor demo applications such as adc_basic, adc_burst, adc_dma, and adc_interrupt. Otherwise, it is not necessary to select this option. Then, click **Finish**.

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

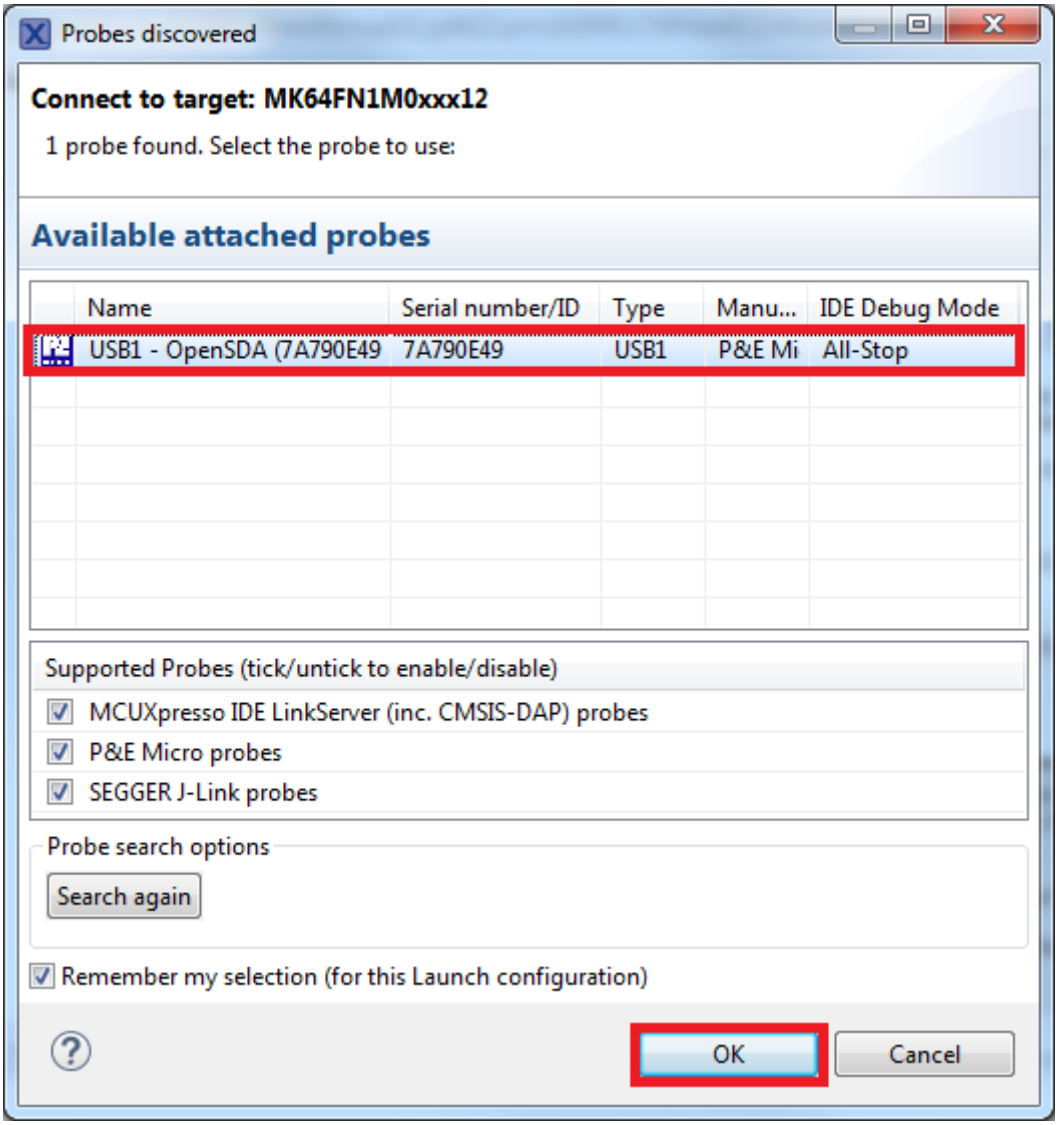
3. 8 data bits



4. 1 stop bit

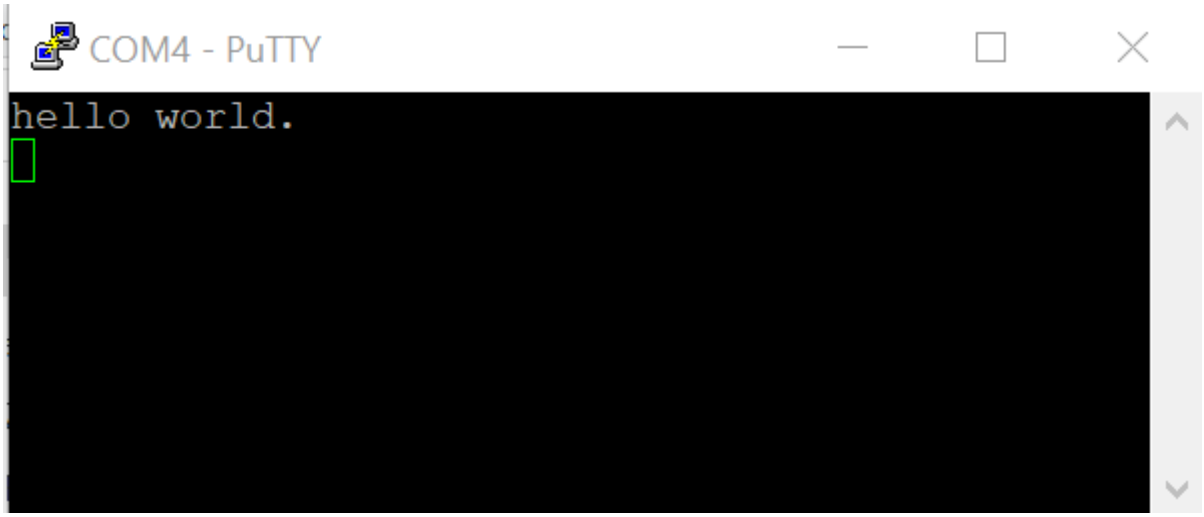
4. On the **Quickstart Panel**, click **Debug** to launch the debug session.

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.)



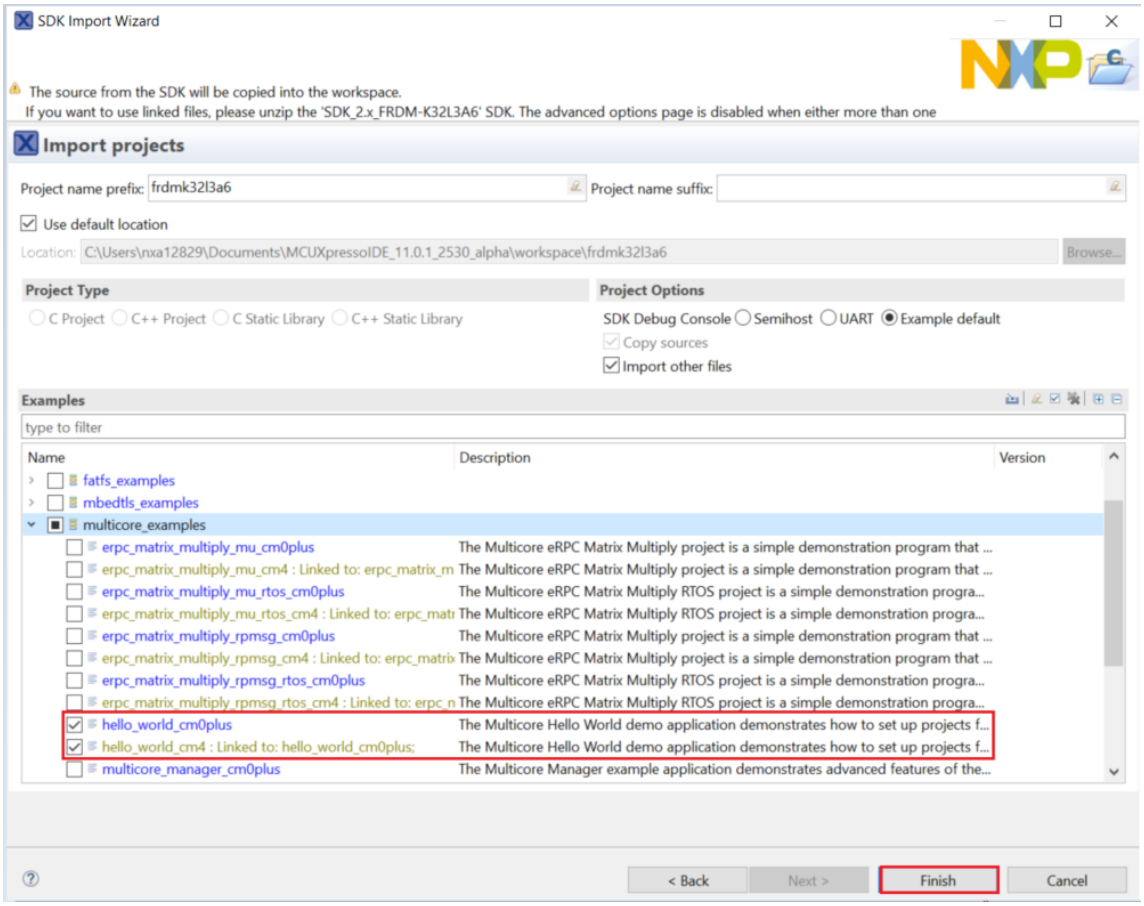
6. The application is downloaded to the target and automatically runs to main().
7. Start the application by clicking **Resume**.

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

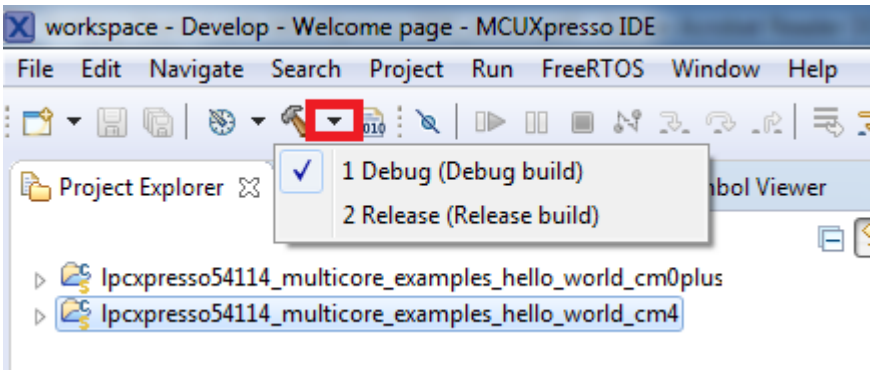


Build a multicore example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world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

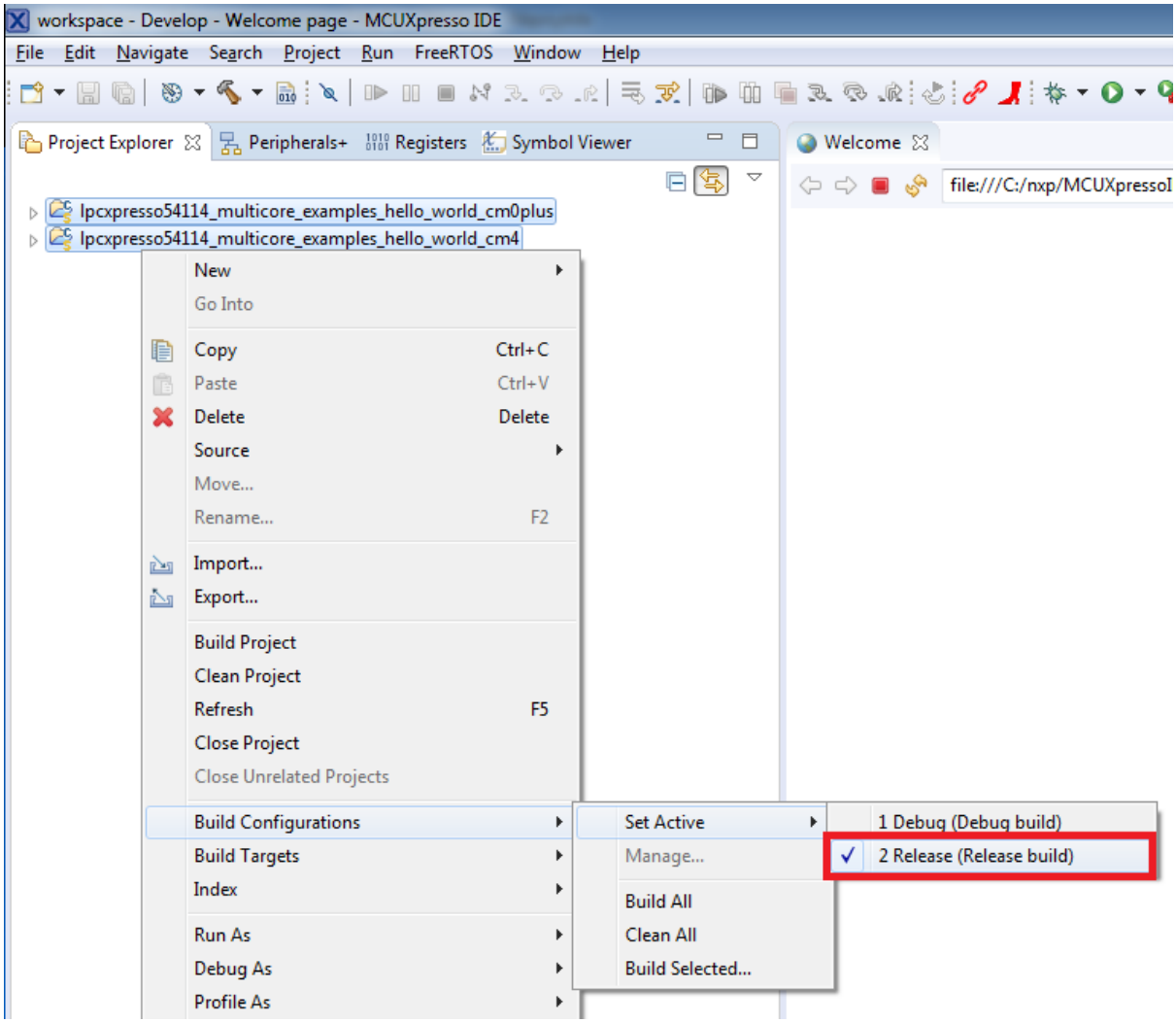


3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the lpcxpresso54114_multicore_examples_hello_world_cm4 project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by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

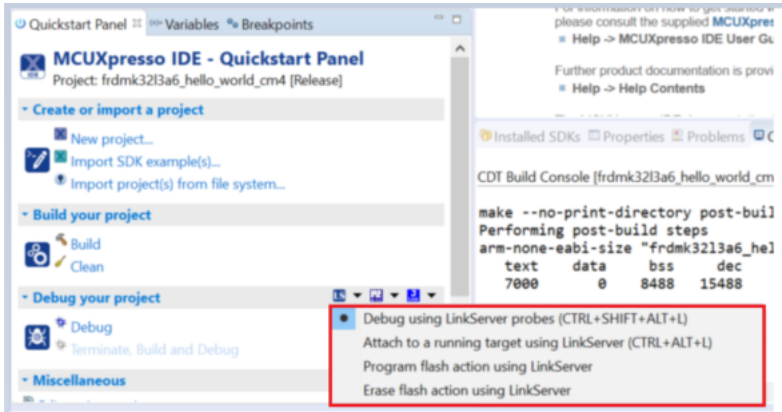


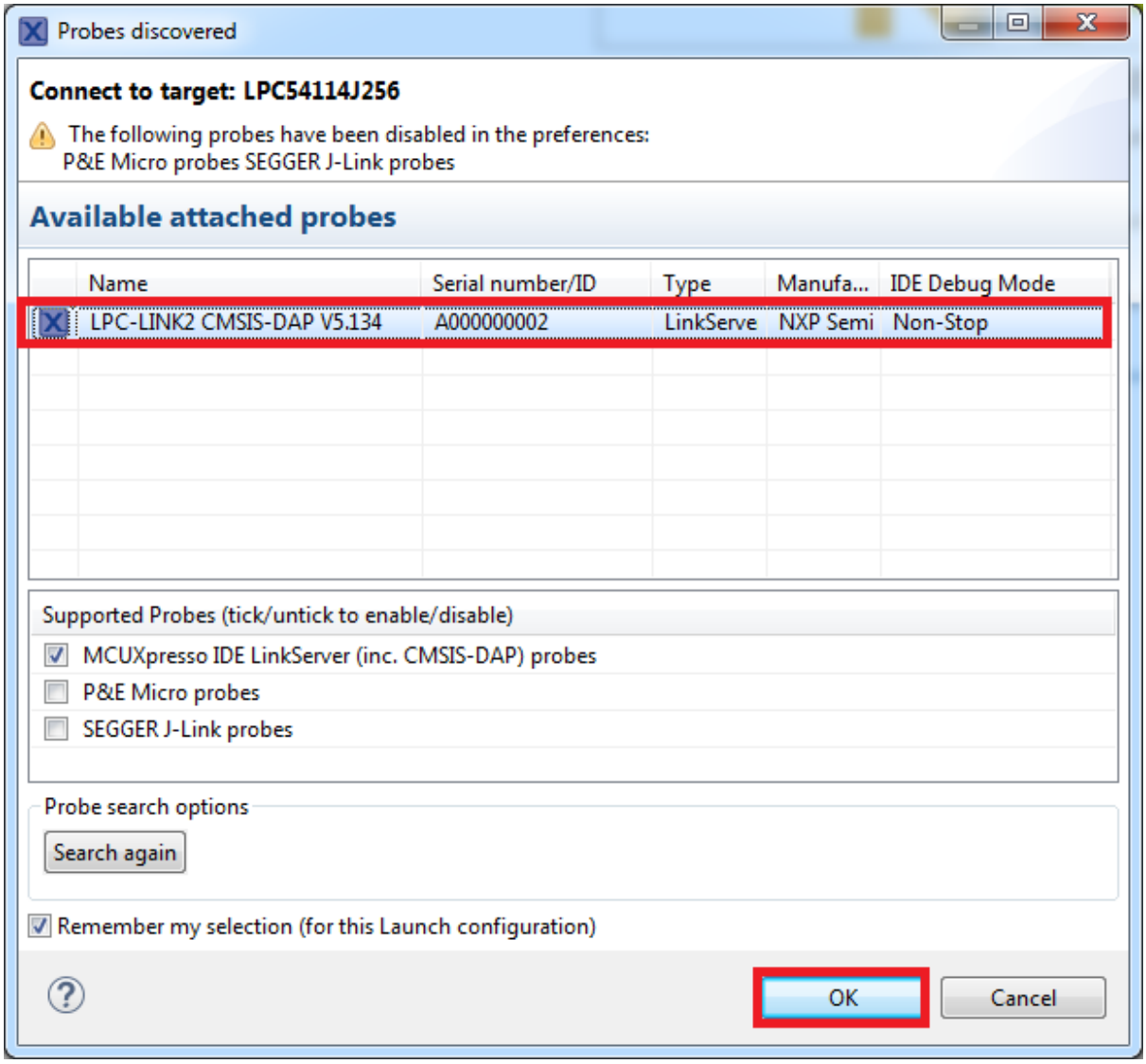
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

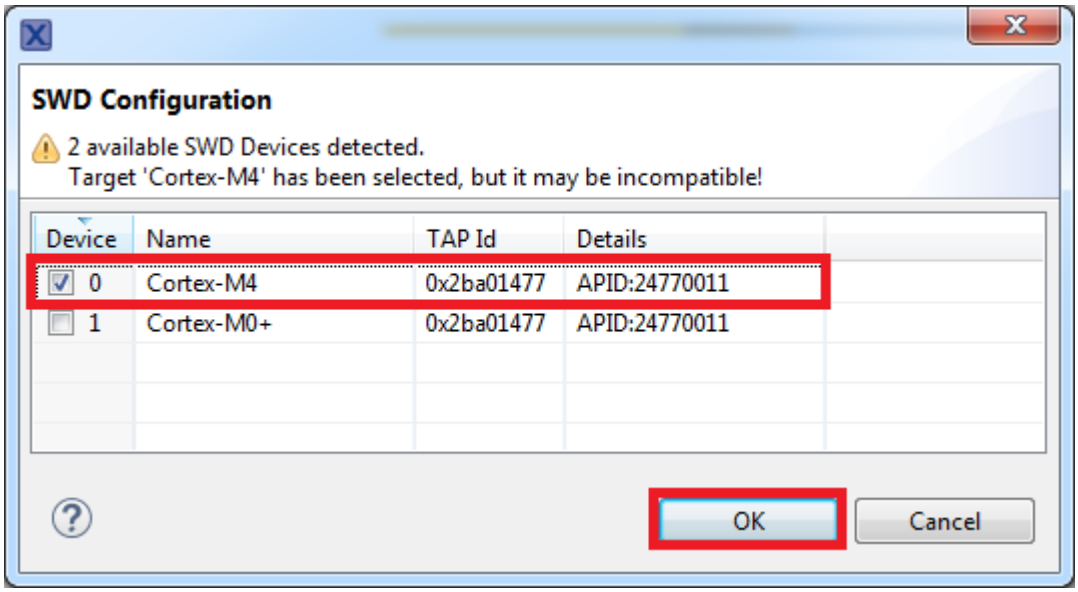
Note: When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build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

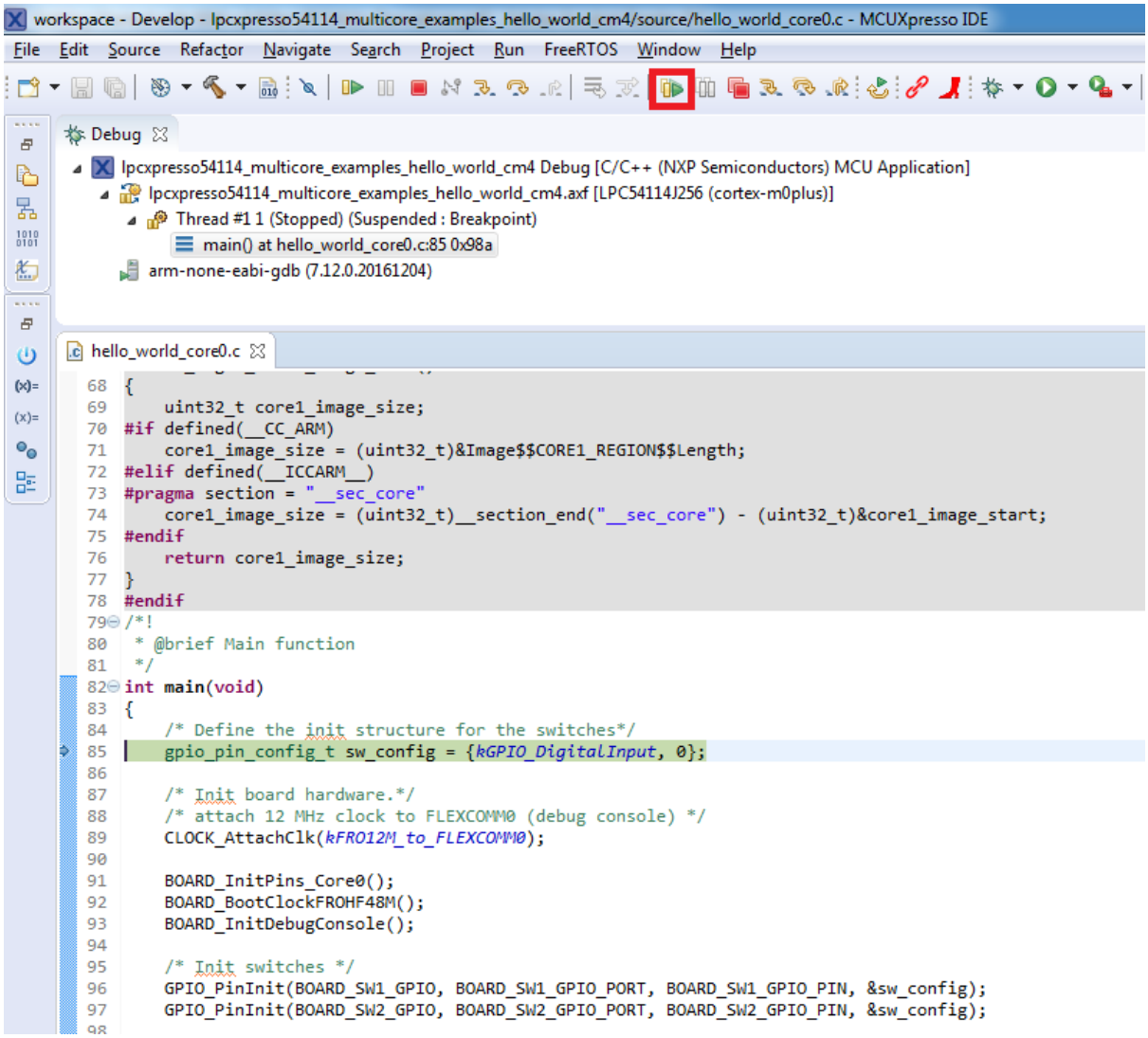


Run a multicore example application The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run an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

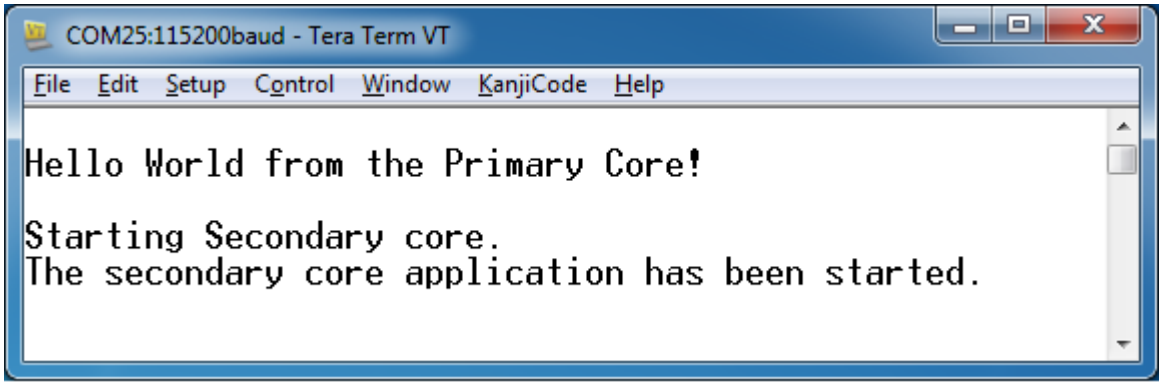



Probes discovered
Connect to target: LPC54114J256
The following probes have been disabled in the preferences:
P&E Micro probes SEGGER J-Link probes
Available attached probes
Name
Serial number/ID
Type
Manufa...
IDE Debug Mode
LPC-LINK2 CMSIS-DAP V5.134
A000000002
LinkServe
NXP Semi
Non-Stop
Supported Probes (tick/untick to enable/disable)
MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
P&E Micro probes
SEGGER J-Link probes
Probe search options
Search again
Remember my selection (for this Launch configuration)
OK
Cancel


SWD Configuration
2 available SWD Devices detected.
Target 'Cortex-M4' has been selected, but it may be incompatible!
Device
Name
TAP Id
Details
0
Cortex-M4
0x2ba01477
APID:24770011
1
Cortex-M0+
0x2ba01477
APID:24770011
OK
Cancel

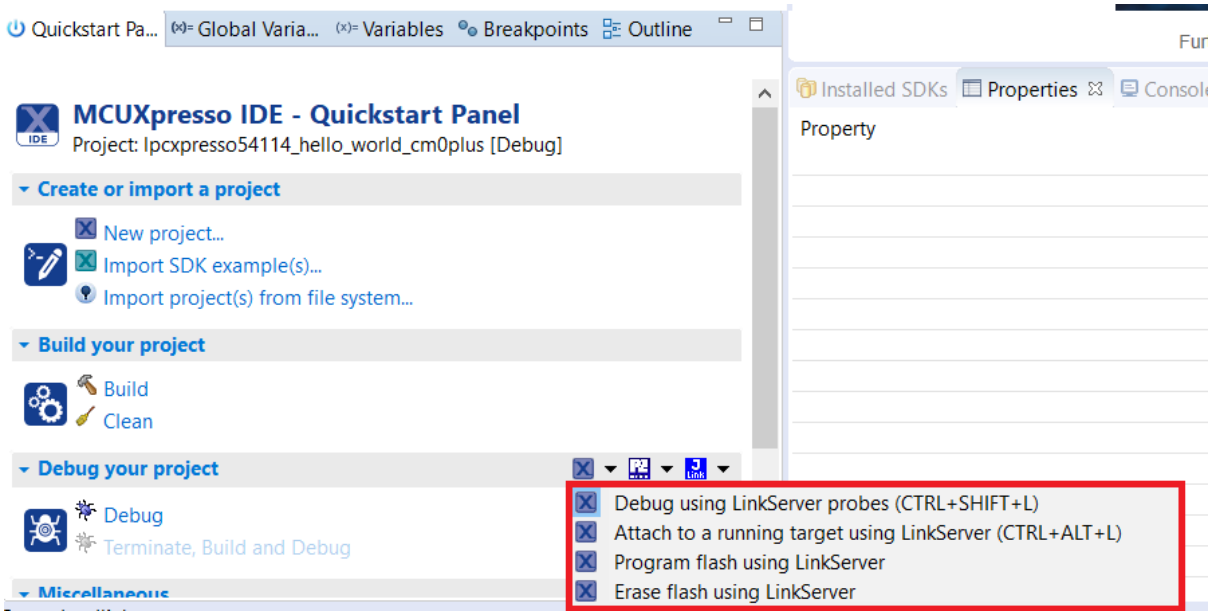


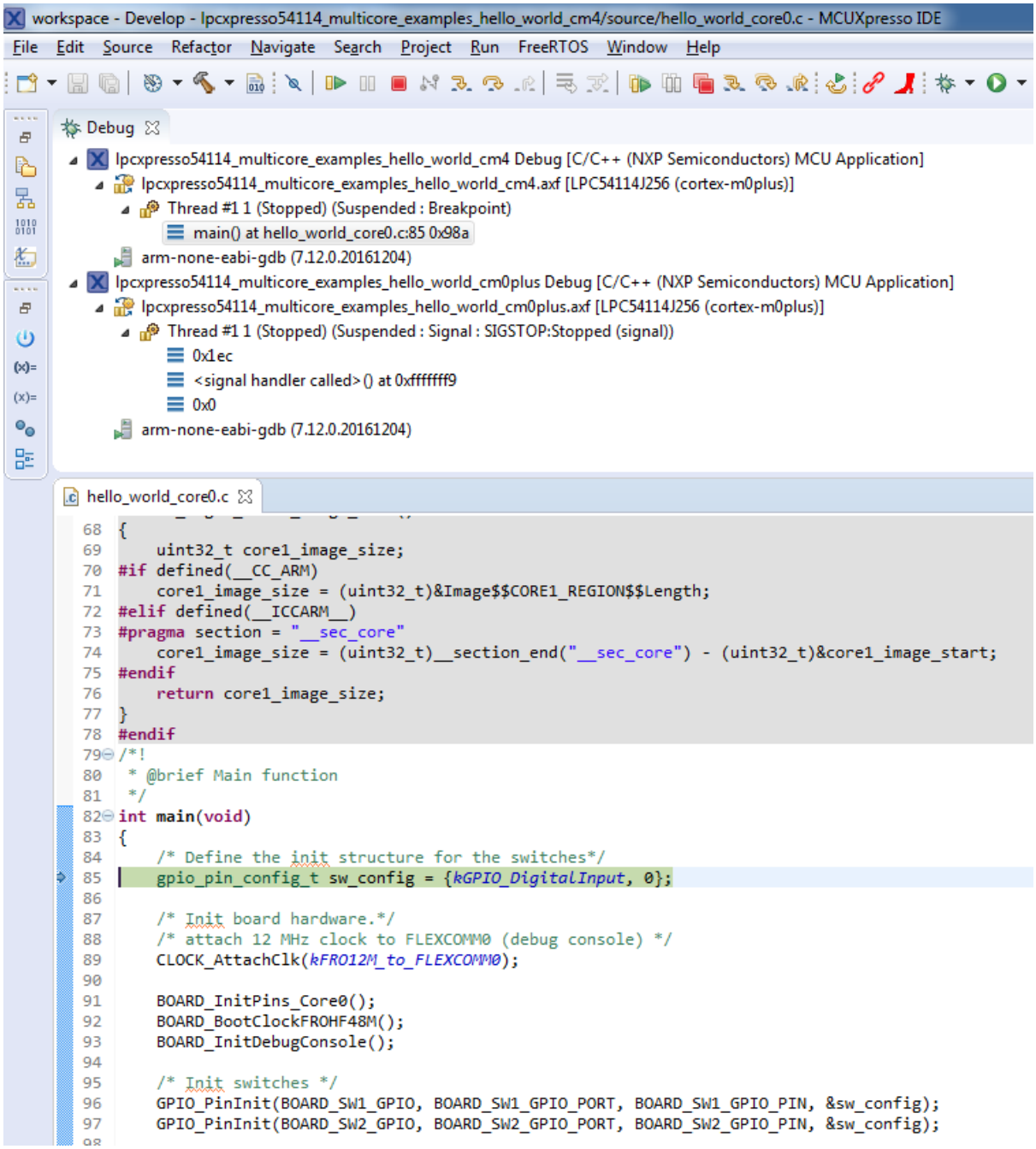
After clicking the “Resume All Debug sessions” button, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



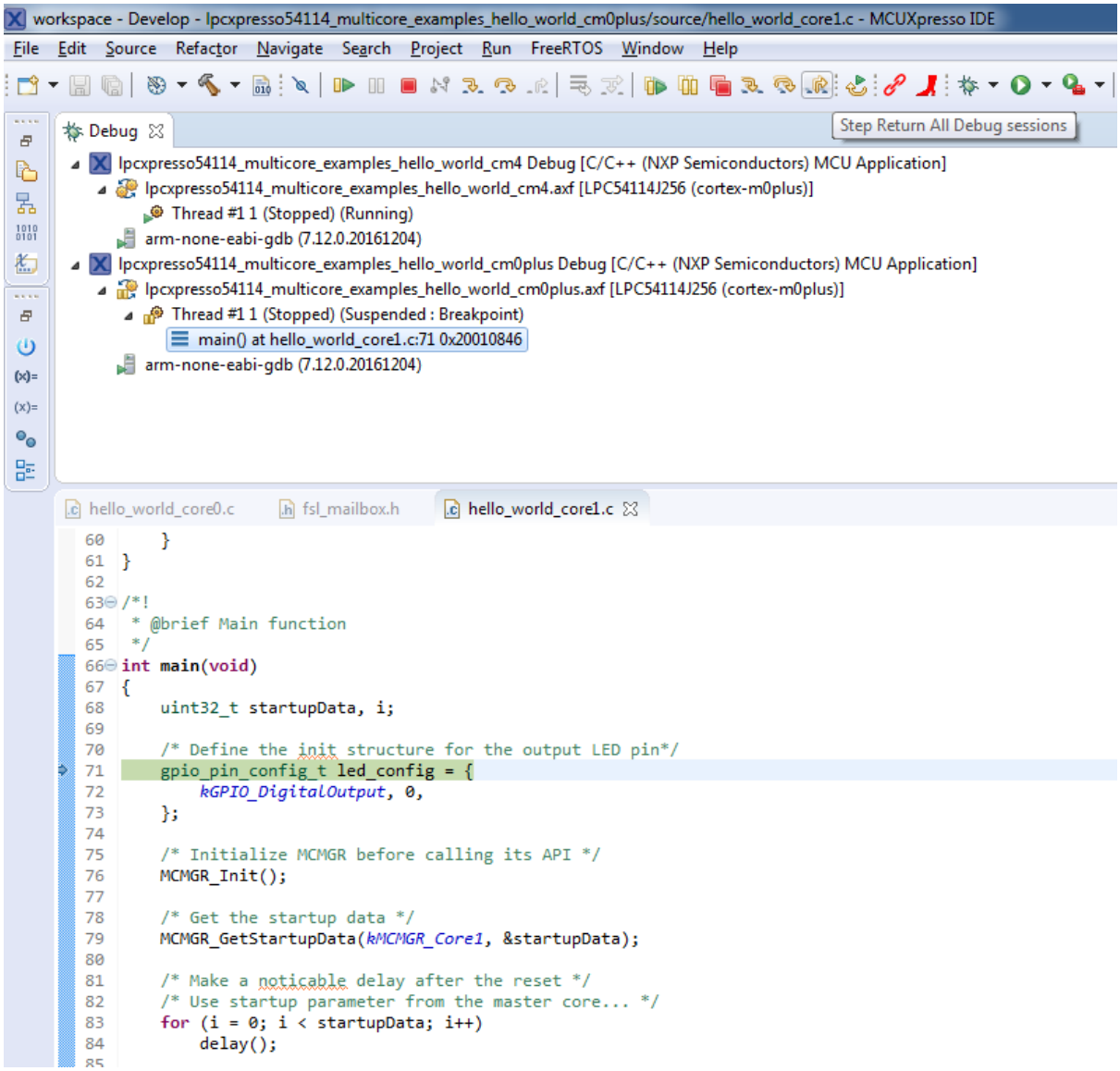
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps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

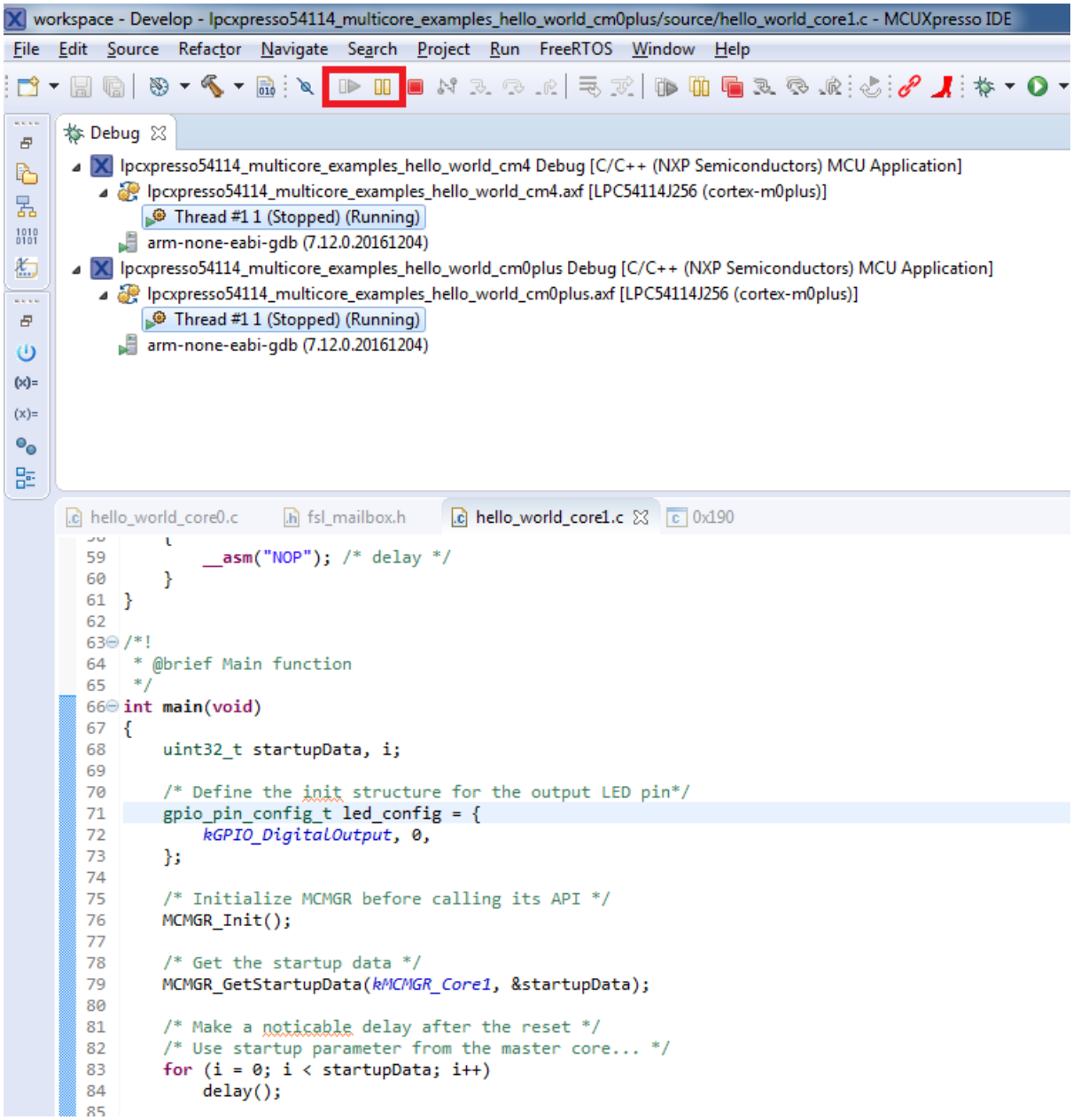

Quickstart Pa...
Global Varia...
Variables
Breakpoints
Outline
MCUXpresso IDE - Quickstart Panel
Project: lpcxpresso54114_hello_world_cm0plus [Debug]
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...
Build your project
Build
Clean
Debug your project
Debug
Terminate, Build and Debug
Miscellaneous
Installed SDKs
Properties
Property
Debug using LinkServer probes (CTRL+SHIFT+L)
Attach to a running target using LinkServer (CTRL+ALT+L)
Program flash using LinkServer
Erase flash using LinkServer

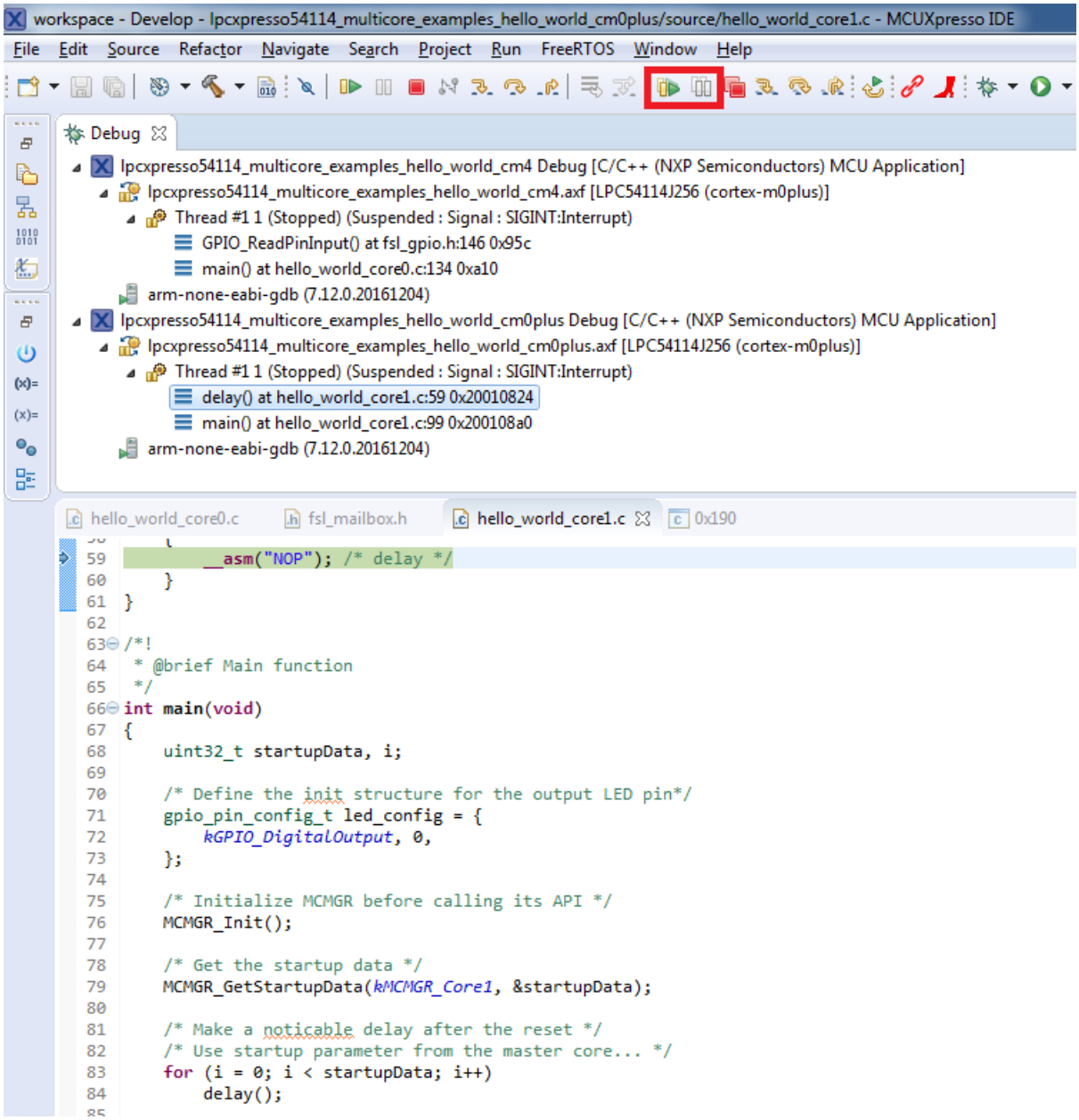


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello_world multicore application then starts running. The primary core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



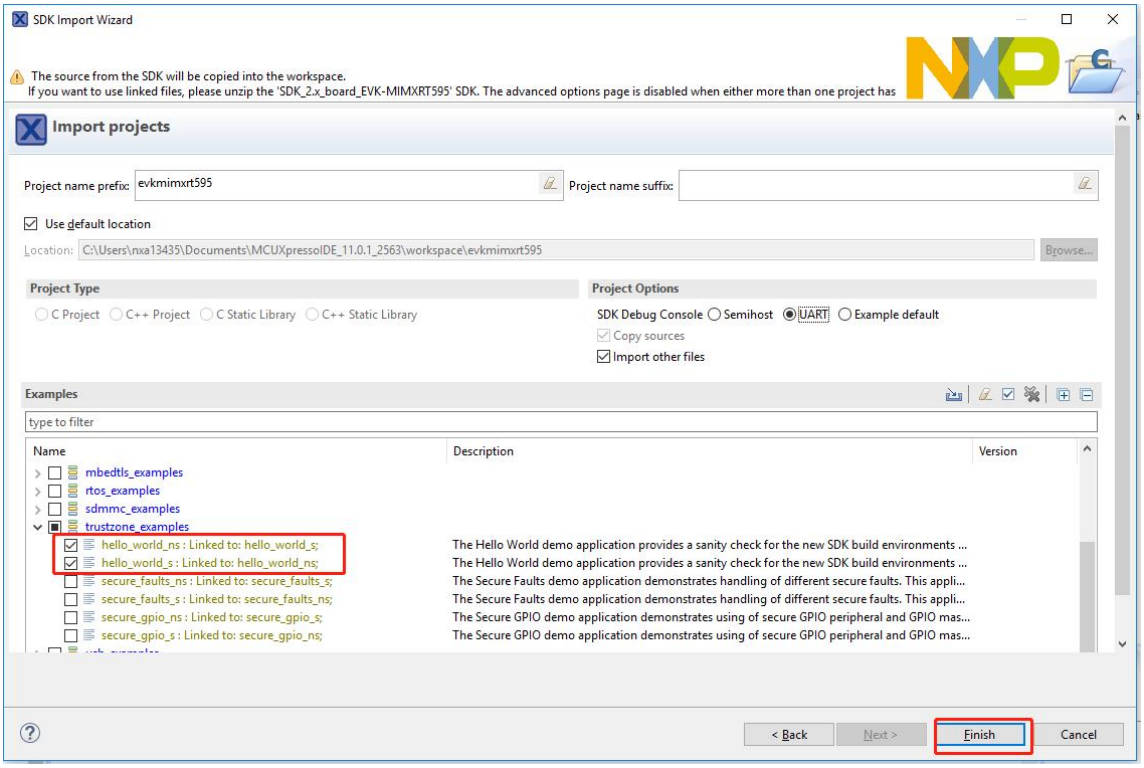
At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both the cores. This is done either by selecting both opened debug sessions (multiple selections) and clicking the “Suspend” / “Resume” control button,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


workspace - Develop - lpcxpresso54114_multicore_examples_hello_world_cm0plus/source/hello_world_core1.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project Run FreeRTOS Window Help
Debug
lpcxpresso54114_multicore_examples_hello_world_cm4 Debug [C/C++ (NXP Semiconductors) MCU Application]
lpcxpresso54114_multicore_examples_hello_world_cm4.axf [LPC54114J256 (cortex-m0plus)]
Thread #1 1 (Stopped) (Running)
arm-none-eabi-gdb (7.12.0.20161204)
lpcxpresso54114_multicore_examples_hello_world_cm0plus Debug [C/C++ (NXP Semiconductors) MCU Application]
lpcxpresso54114_multicore_examples_hello_world_cm0plus.axf [LPC54114J256 (cortex-m0plus)]
Thread #1 1 (Stopped) (Running)
arm-none-eabi-gdb (7.12.0.20161204)
hello_world_core0.c
fsl_mailbox.h
hello_world_core1.c
0x190
59 __asm("NOP"); /* delay */
60 }
61 }
62
63 /*!
64 * @brief Main function
65 */
66 int main(void)
67 {
68 uint32_t startupData, i;
69
70 /* Define the init structure for the output LED pin*/
71 gpio_pin_config_t led_config = {
72 kGPIO_DigitalOutput, 0,
73 };
74
75 /* Initialize MCMGR before calling its API */
76 MCMGR_Init();
77
78 /* Get the startup data */
79 MCMGR_GetStartupData(kMCMGR_Core1, &startupData);
80
81 /* Make a noticable delay after the reset */
82 /* Use startup parameter from the master core... */
83 for (i = 0; i < startupData; i++)
84 delay();
85

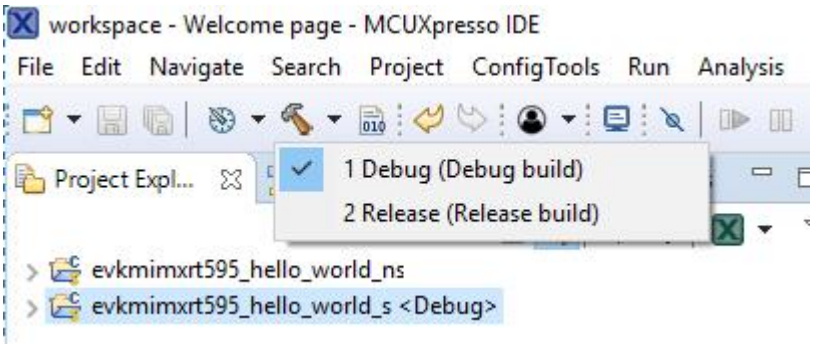


Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the hello_world example application targeted for the MIMXRT595-EVK hardware platform is used as an example, though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

1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for MIMXRT595-EVK is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **MIMXRT500** folder and select **MIMXRT595S**. Then, select **evkmimxrt595** and click **Next**.
2. Expand the trustzone_examples/ folder and select hello_world_s. Because TrustZone examples are linked together, the non-secure project is automatically imported with the secure project, and there is no need to select it explicitly. Then, click **Finish**.

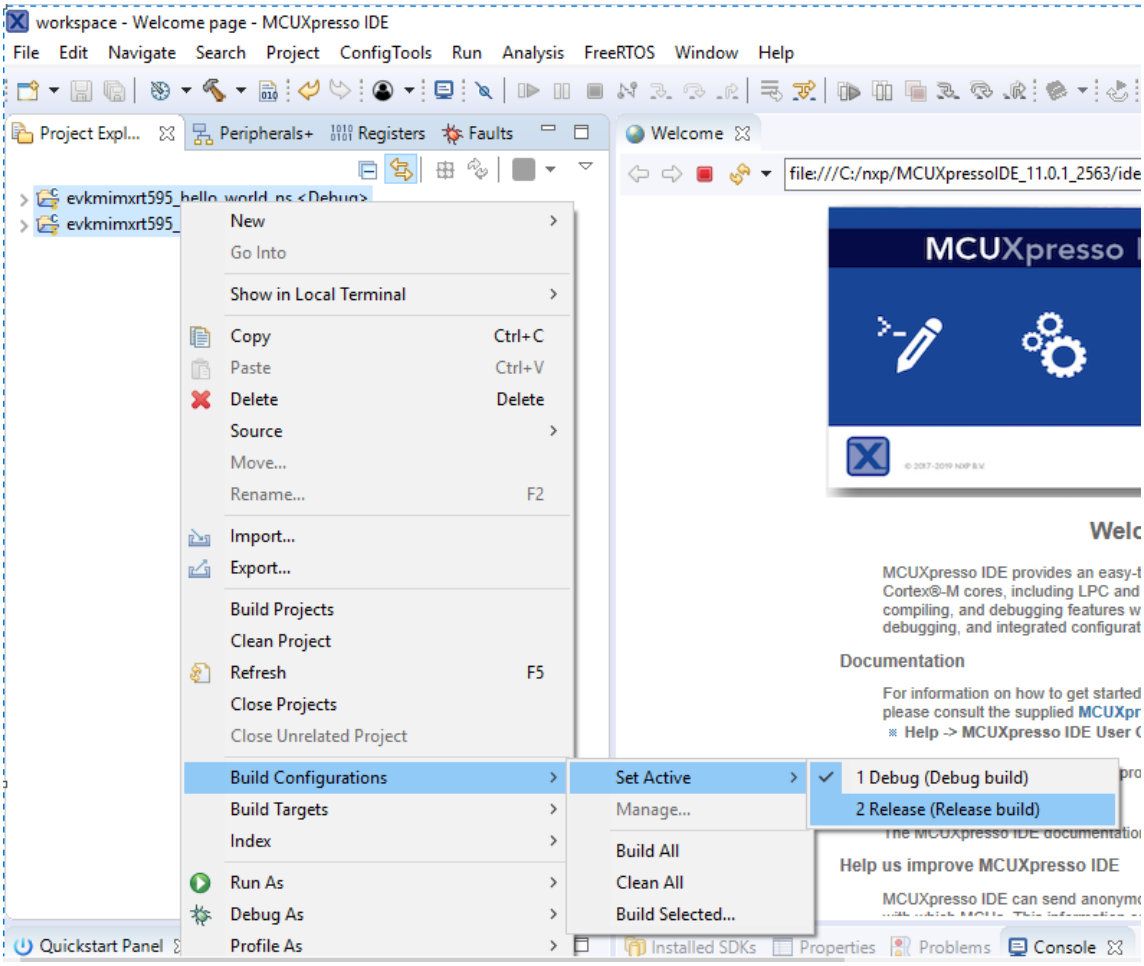


3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the evkmimxrt595_hello_world_s project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward facing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



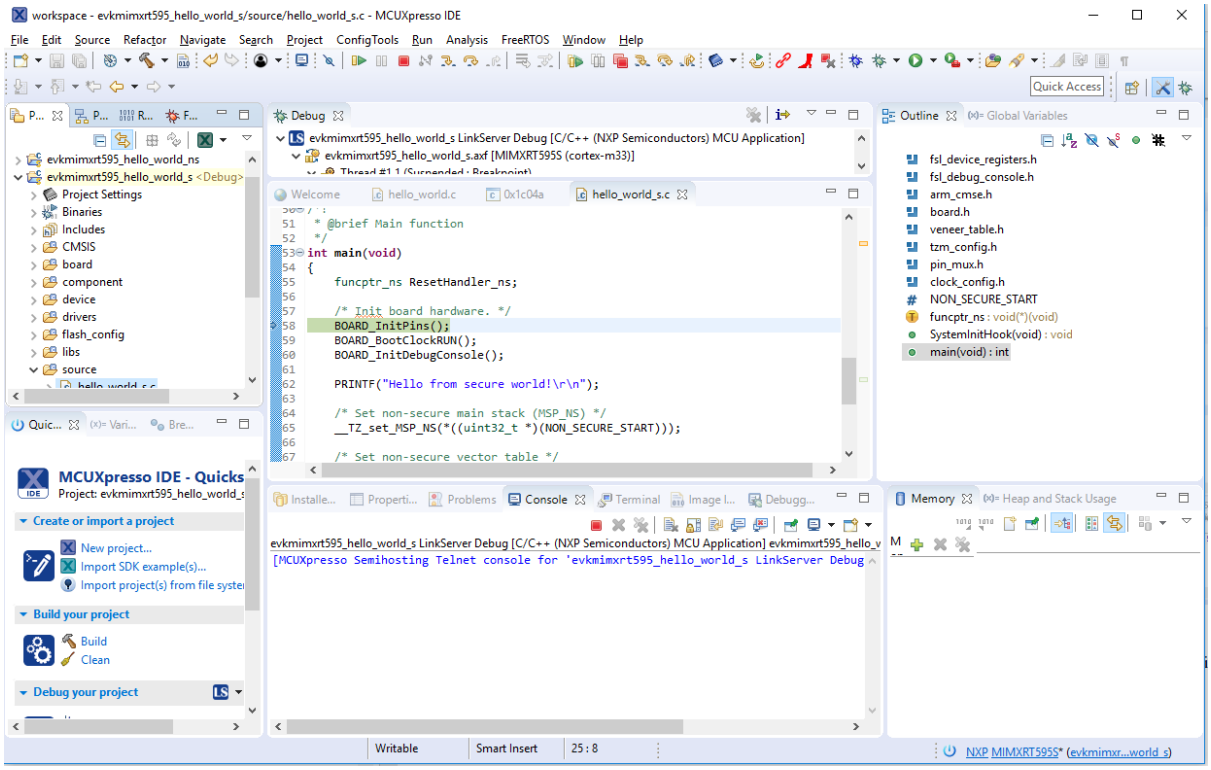
The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when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

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



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

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



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

Run a demo application using IAR

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

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

Build an example application

Do the following steps to build the hello_world example application.

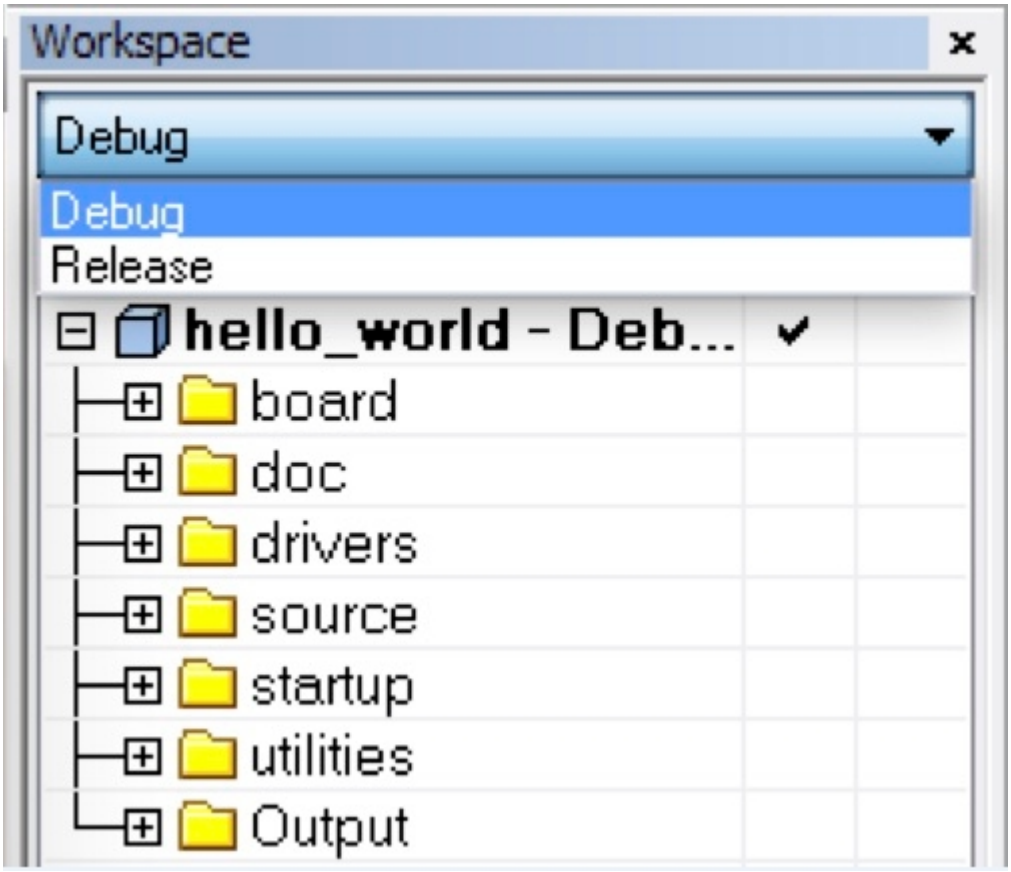
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

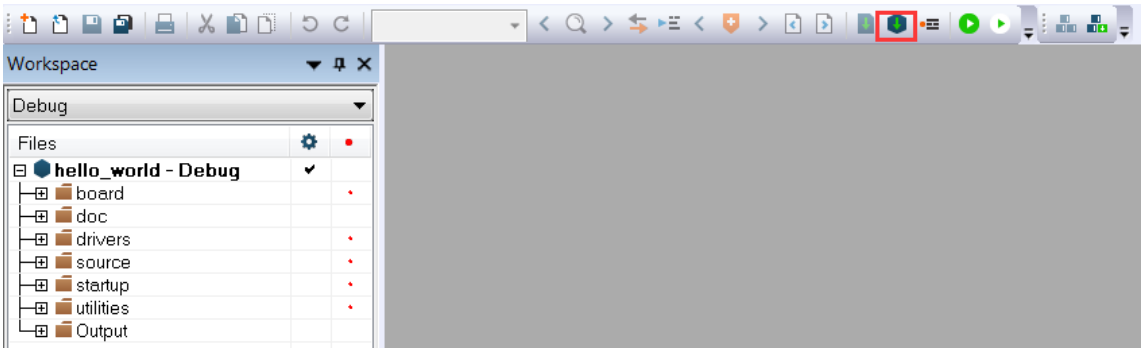
   Other example applications may have additional folders in their path.

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

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



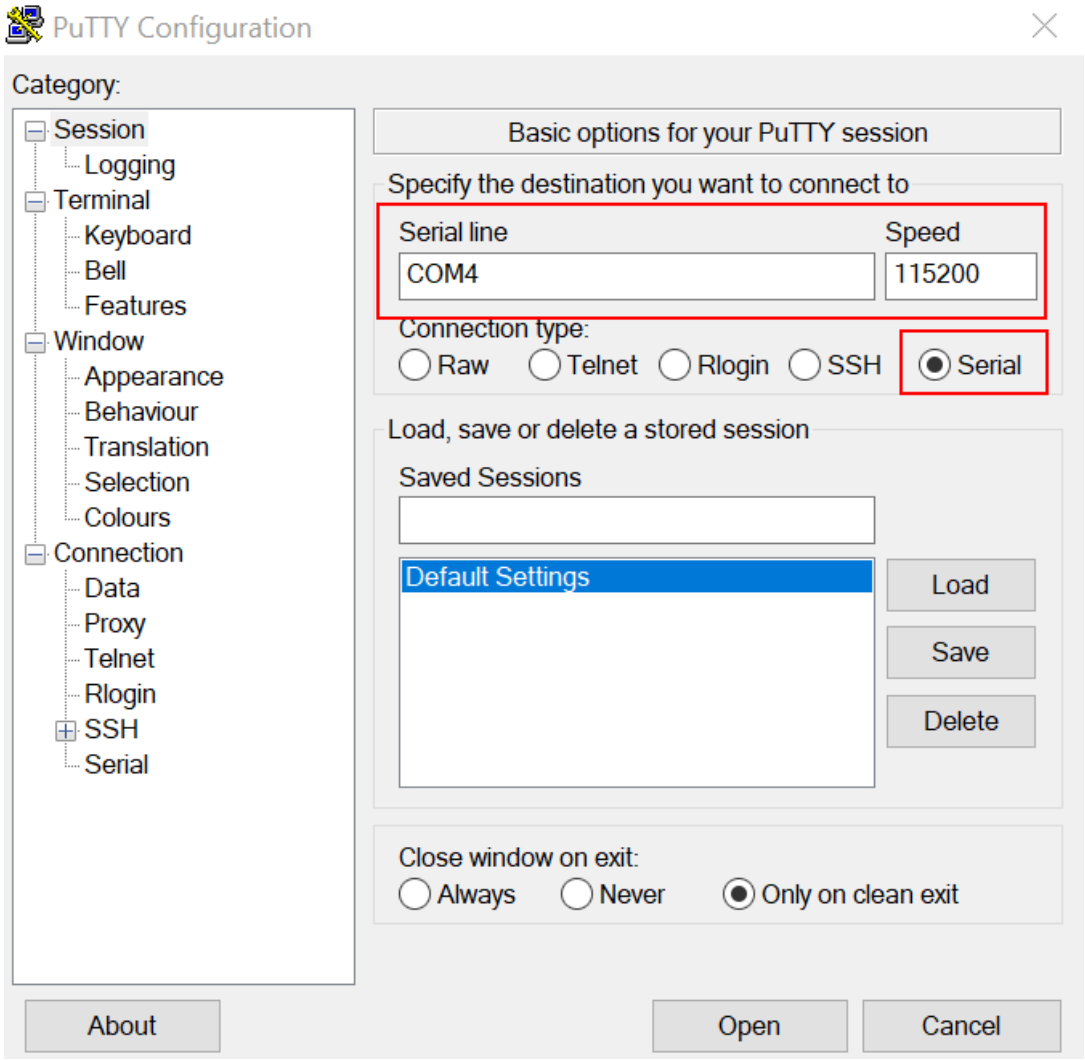
3. To build the demo application, click **Make**, highlighted in red in following figure.



4. The build completes without errors.

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

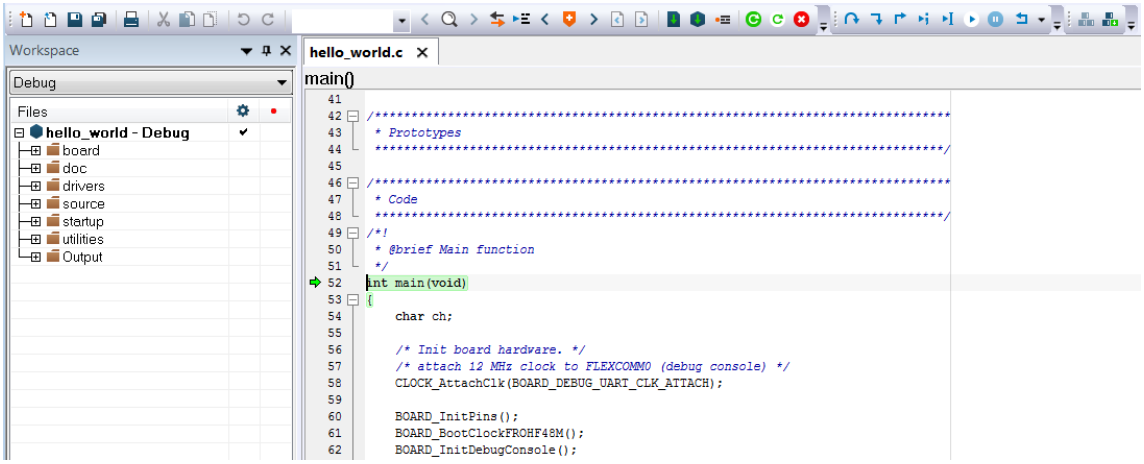
1. Ensure the host driver for the debugger firmware has been installed. See *On-board debugger*.
2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
    1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
    2. No parity
    3. 8 data bits



4. 1 stop bit

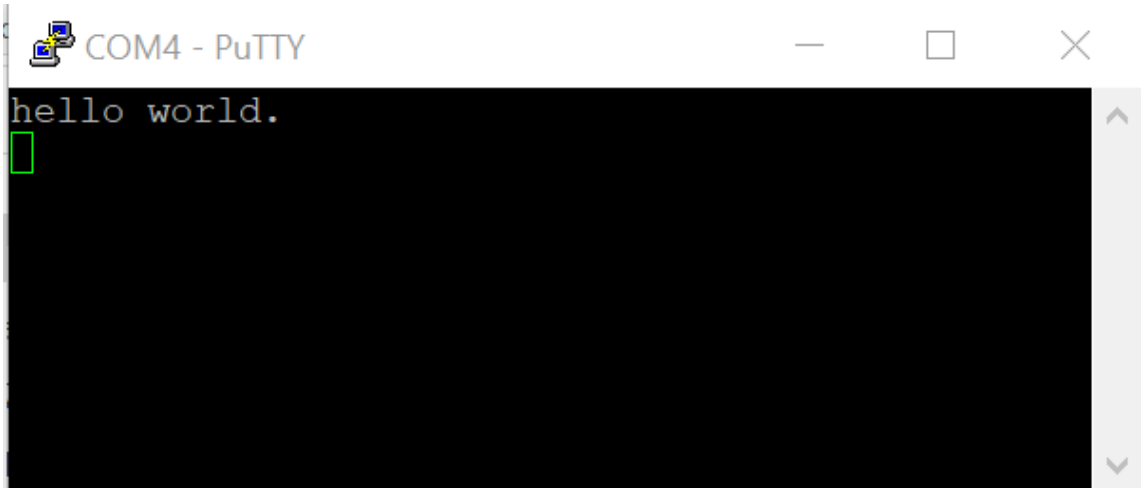
4. In IAR, click the **Download and Debug** button to download the application to the target.

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



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

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



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

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

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

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

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

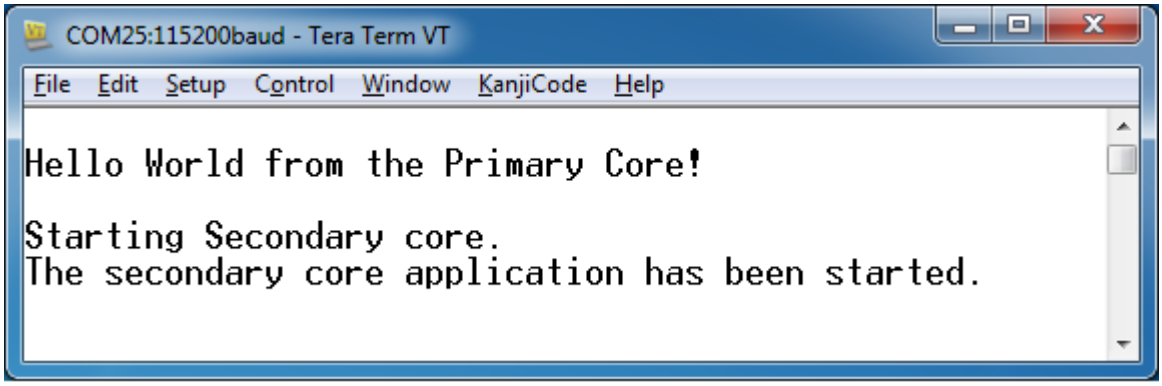
Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm0plus) first, because the primary core application project (cm4) must know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker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

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

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

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

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



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

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

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

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

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

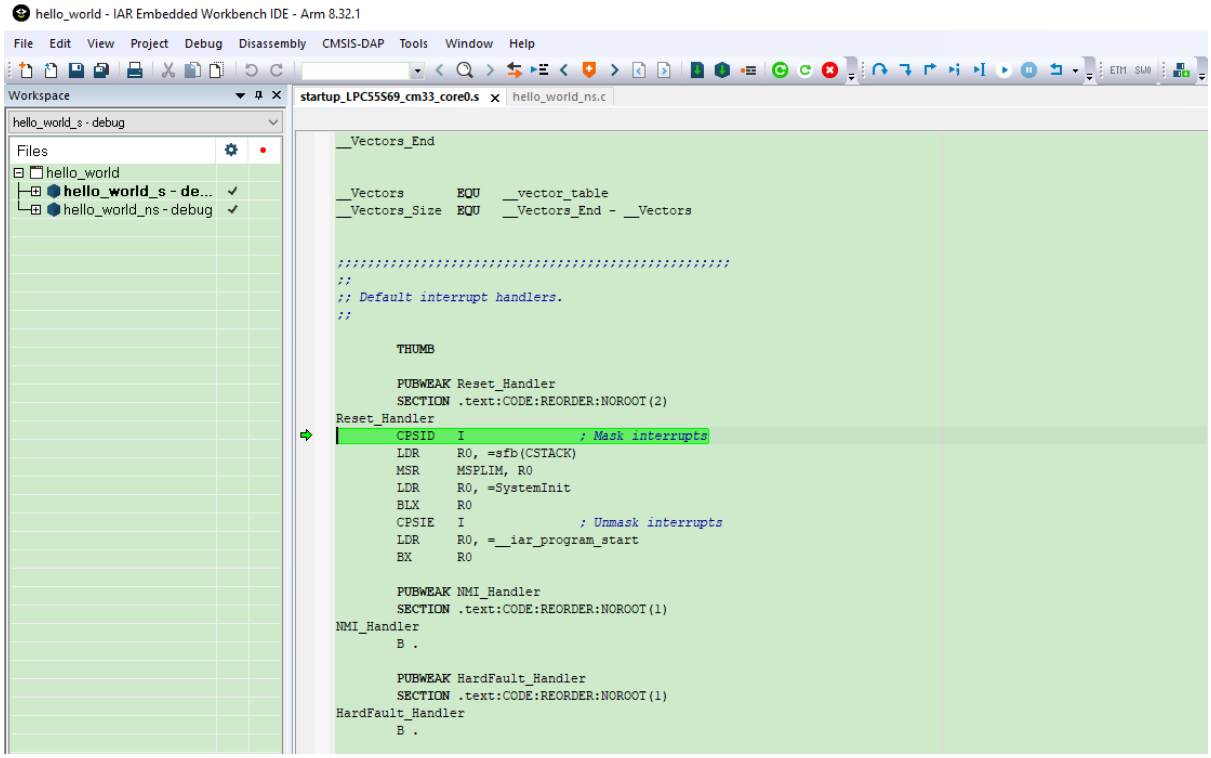
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_
↪ns.eww
```

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

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

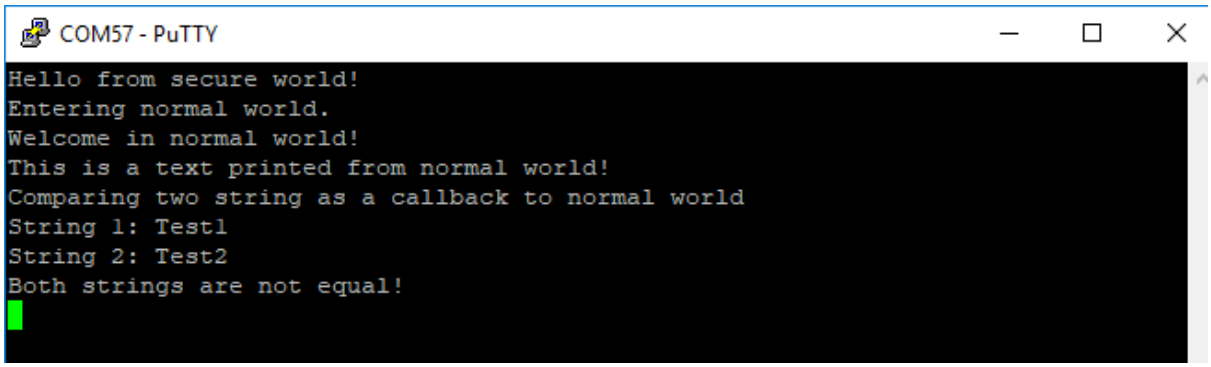
This project hello_world.eww contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because the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

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

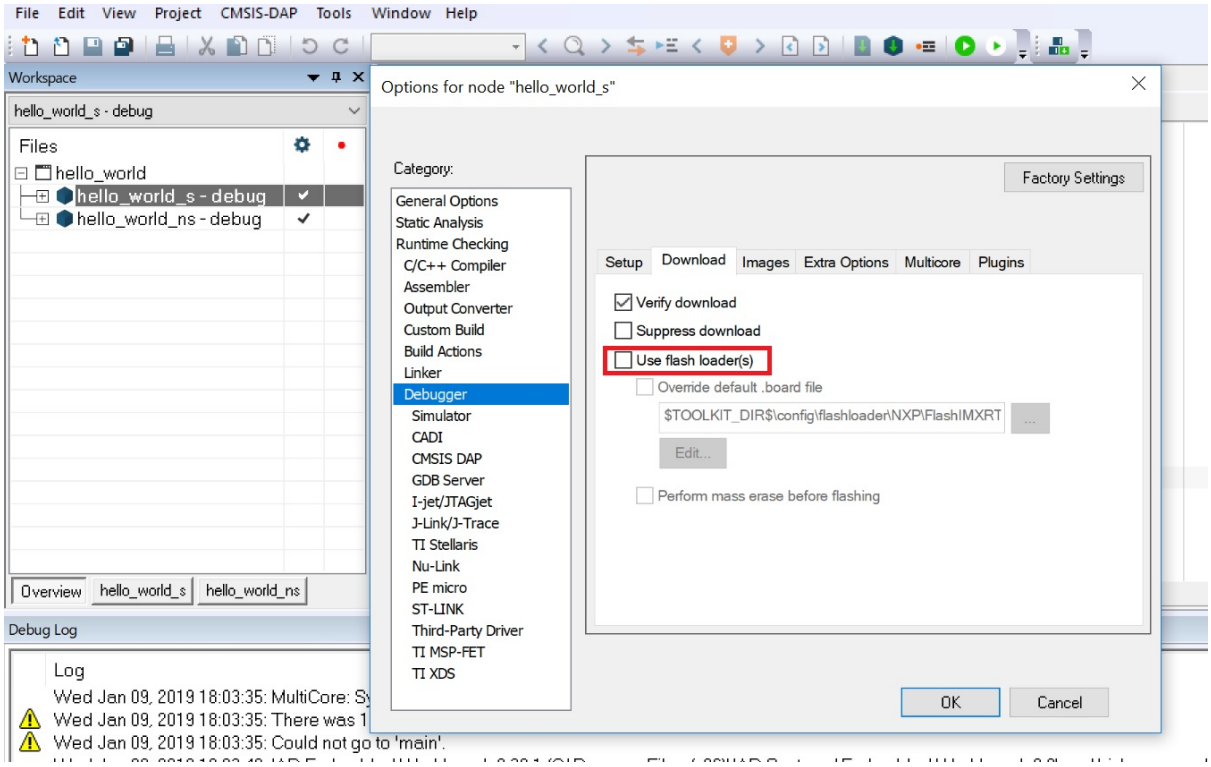


Run the code by clicking **Go** to start the application.

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



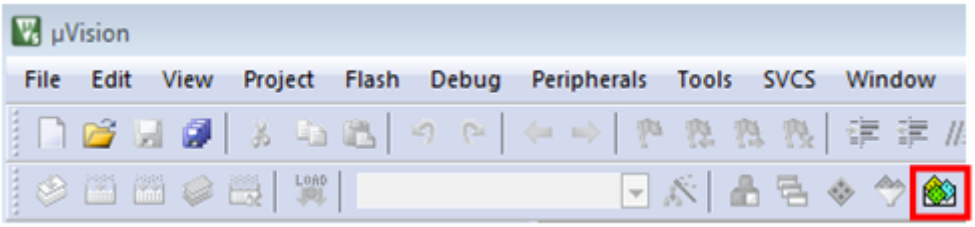
Note: If the application is running in RAM (debug/release build target), in **Options****>****Debugger** > **Download** tab, disable **Use flash loader(s)**. This can avoid the _ns download issue on i.MXRT500.



Run a demo using Keil MDK/µVision This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK.

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

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



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

Build an example application

1. Open the desired example application workspace in:

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

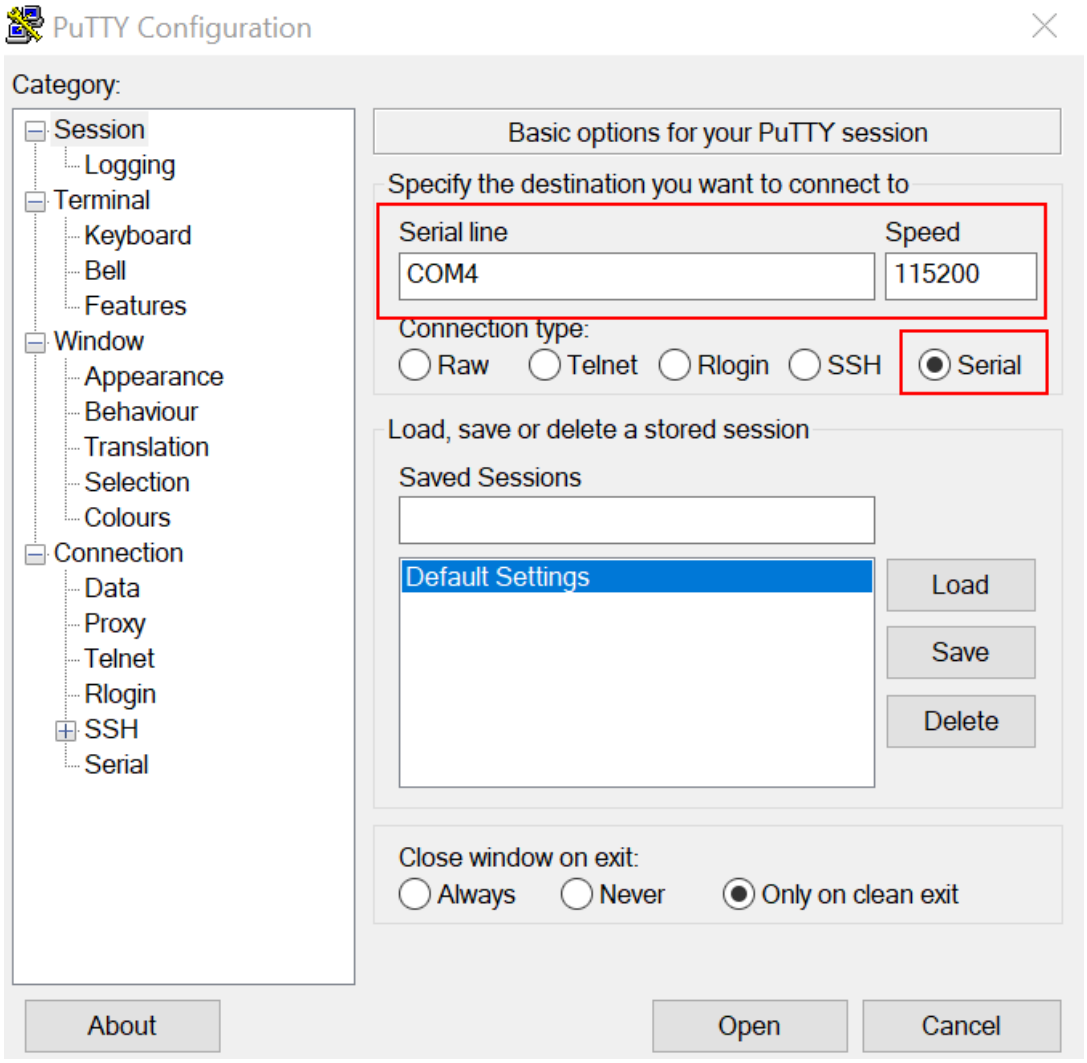
The workspace file is named as <demo_name>.uvmpw. For this specific example, the actual path is:

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

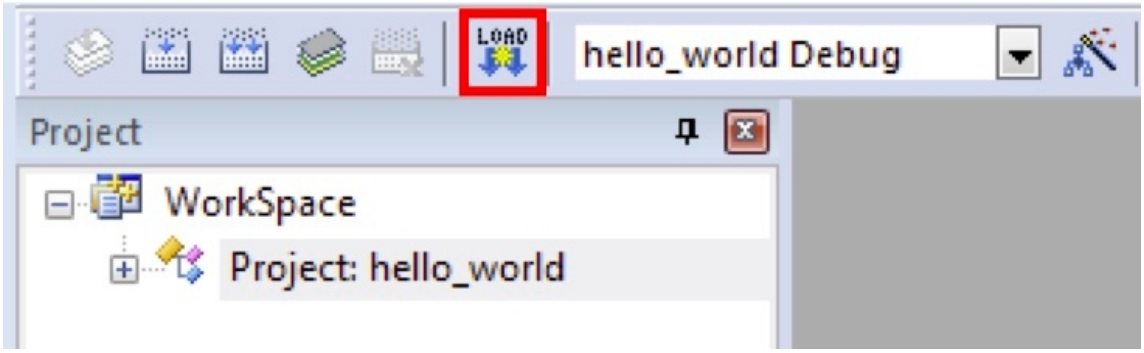
3. The build completes without errors.

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

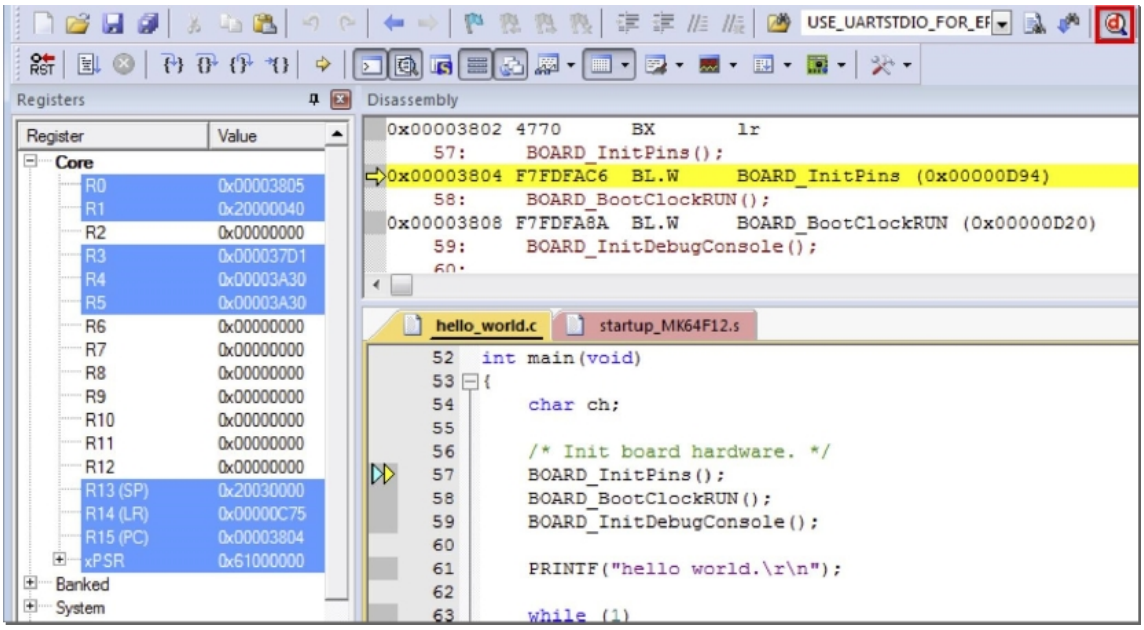
1. Ensure the host driver for the debugger firmware has been installed. See *On-board debugger*.
2. Connect the development platform to your PC via USB cable using USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine the COM port number, see *How to determine COM port*. Configure the terminal with these settings:
    1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
    2. No parity
    3. 8 data bits
    4. 1 stop bit



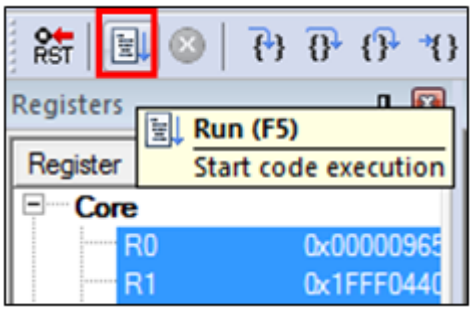
4. In µVision, after the application is built, click the **Download** button to download the application to the target.



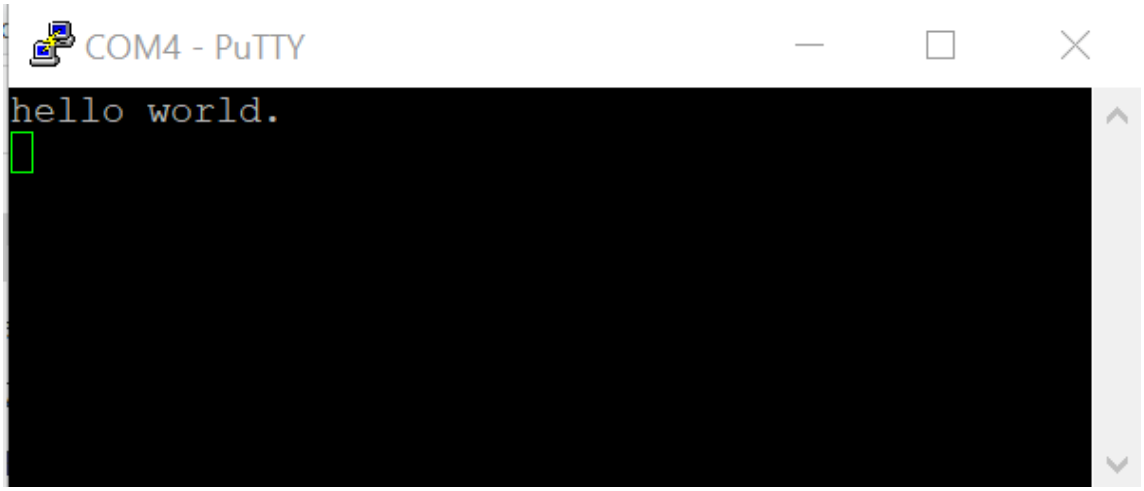
5. After clicking the **Download** button, the application downloads to the target and is running. To debug the application, click the **Start/Stop Debug Session** button, highlighted in red.



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:

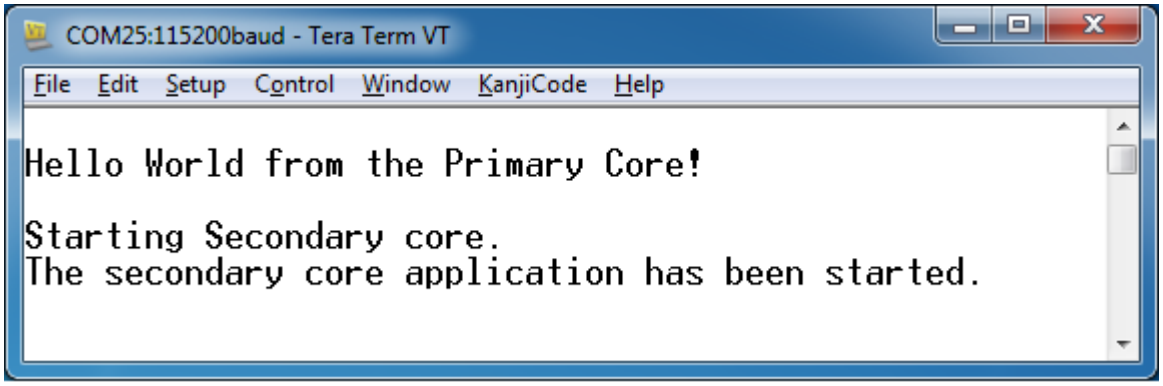
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
↪cm0plus.uvmpw
```

```
<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.

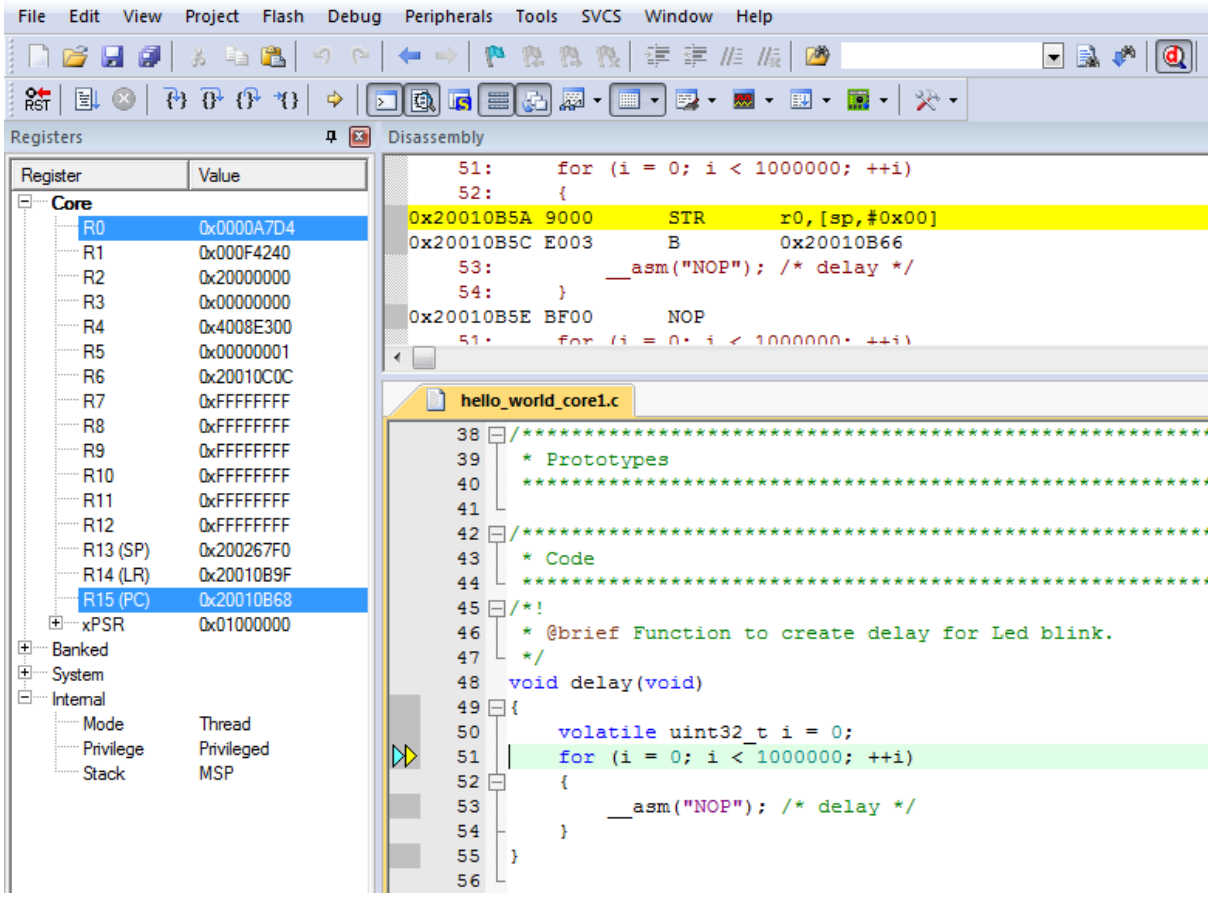
Run a multicore example application The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example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

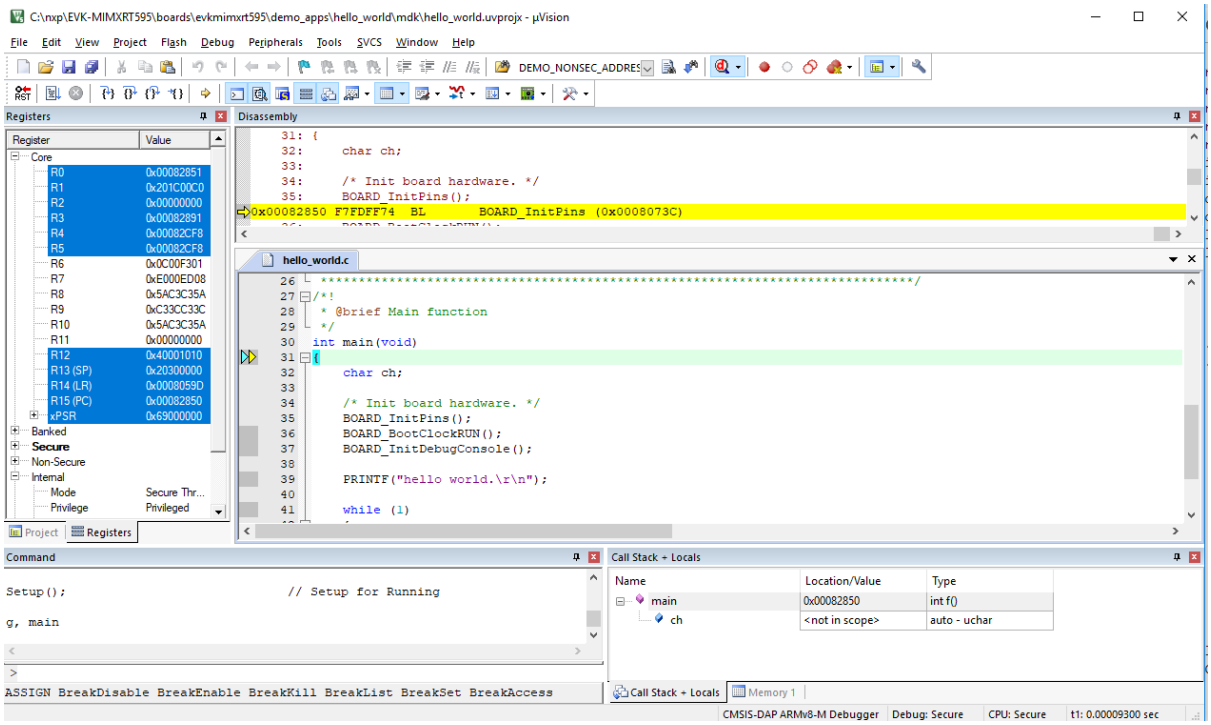
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.
↪uvmpw
```

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.

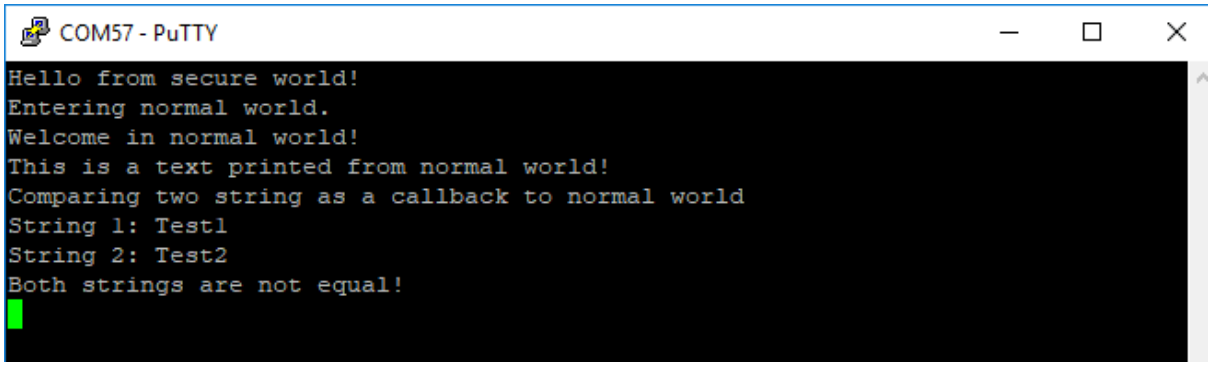
Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

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 µVision. 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:

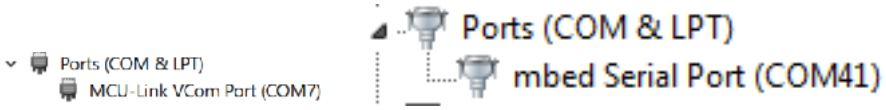
```
$ dmesg | grep "ttyUSB"
  [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
  [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
```

There are two ports, one is for core0 debug console and the other is for core1.

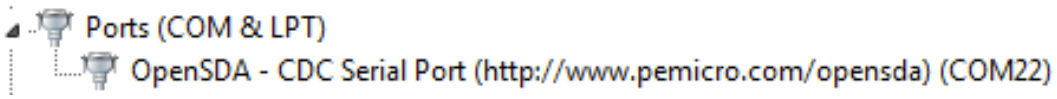
2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

   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.

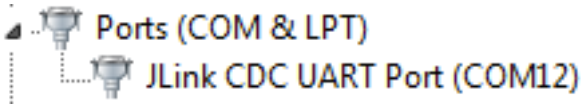
   1. **CMSIS-DAP/mbed/DAPLink** interface:

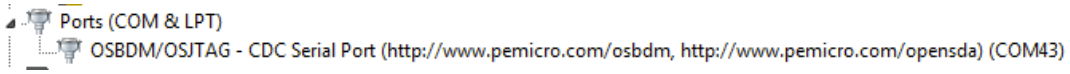
   2. **P&E Micro**:

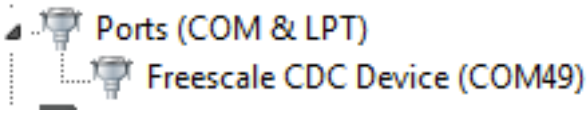
   3. **J-Link**:

   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 LPCScrypt. 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 LPCScrypt 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 LPCScrypt.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScrypt user guide (LPCScrypt, select **LPCScrypt**, 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 LPCScrypt installation directory (<LPCScrypt install dir>).
    1. To program CMSIS-DAP debug firmware: <LPCScrypt install dir>/scripts/program_CMSIS
    2. To program J-Link debug firmware: <LPCScrypt install dir>/scripts/program_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

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

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

See the table in *Default debug interfaces* to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

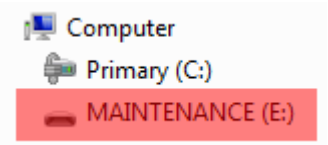
- For boards with CMSIS-DAP firmware, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
- For boards with a P&E Micro interface, see PE micro to download and install the P&E Micro Hardware Interface Drivers package.

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

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

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

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



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

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

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

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

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

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

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

The host driver must be installed before debugging.

- See PE micro to download and install the P&E Micro Hardware Interface Drivers package.

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

The host driver must be installed before debugging.

- See PE micro to download and install the P&E Micro Hardware Interface Drivers package.

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

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

continues on next page

Table 1 – continued from previous page

Hardware platform	Default debugger firmware	On-board debugger probe
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: Dependency-Installation
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 GNU Toolchain	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

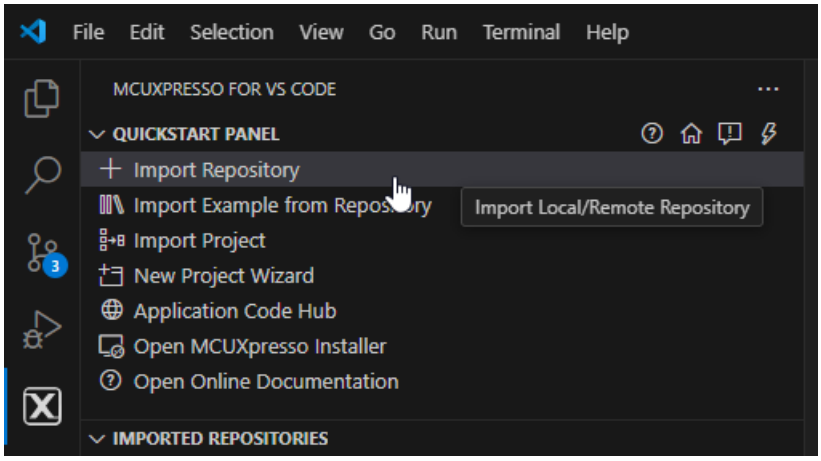
Extension Installation

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 Wiki for detailed installation and setup instructions.

SDK Import and Setup

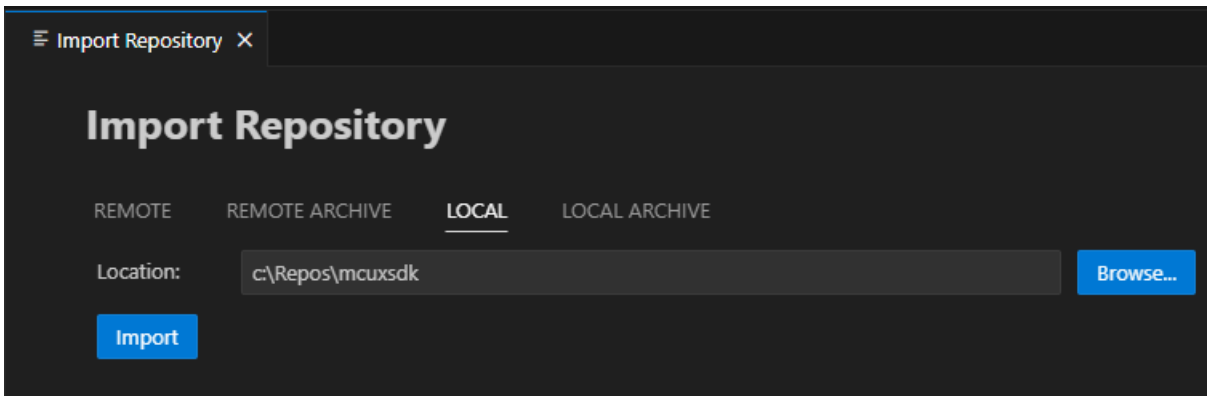
Import Methods The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

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

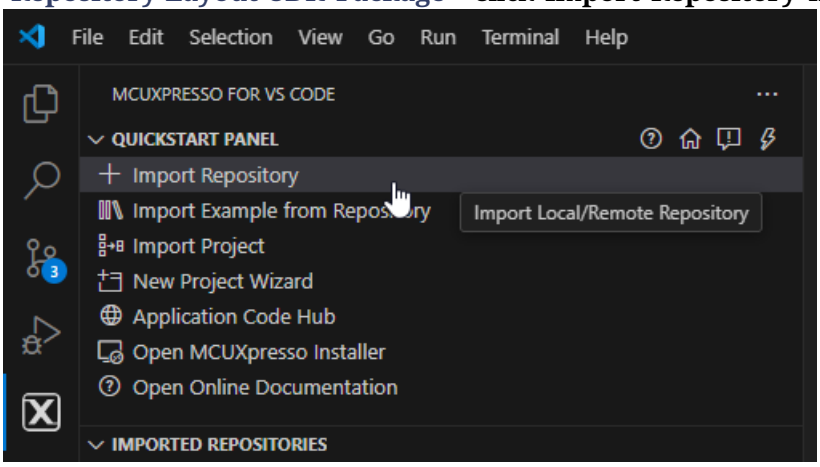


Note: You can import the SDK in several ways. Refer to MCUXpresso for VS Code Wiki for details.

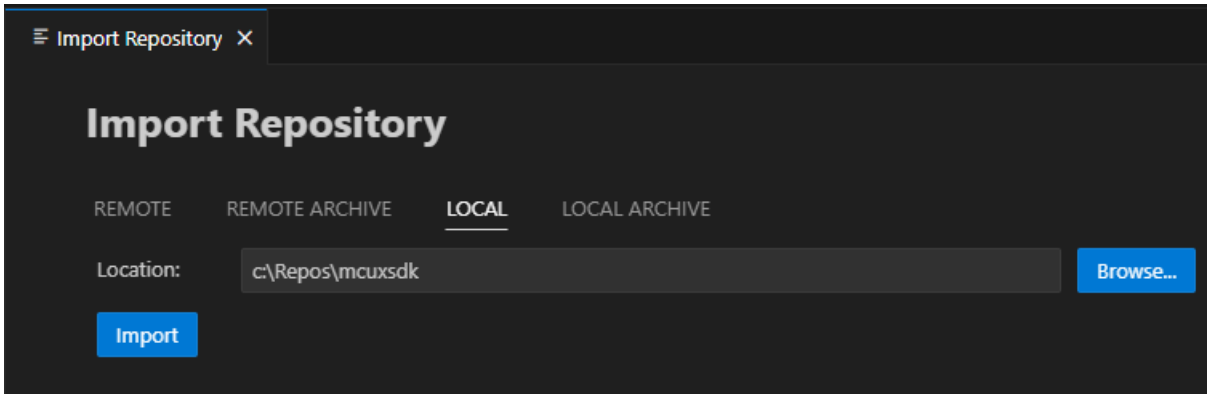
Select **Local** if you've already obtained the SDK according to *setting up the repo*. Select your location and click **Import**.



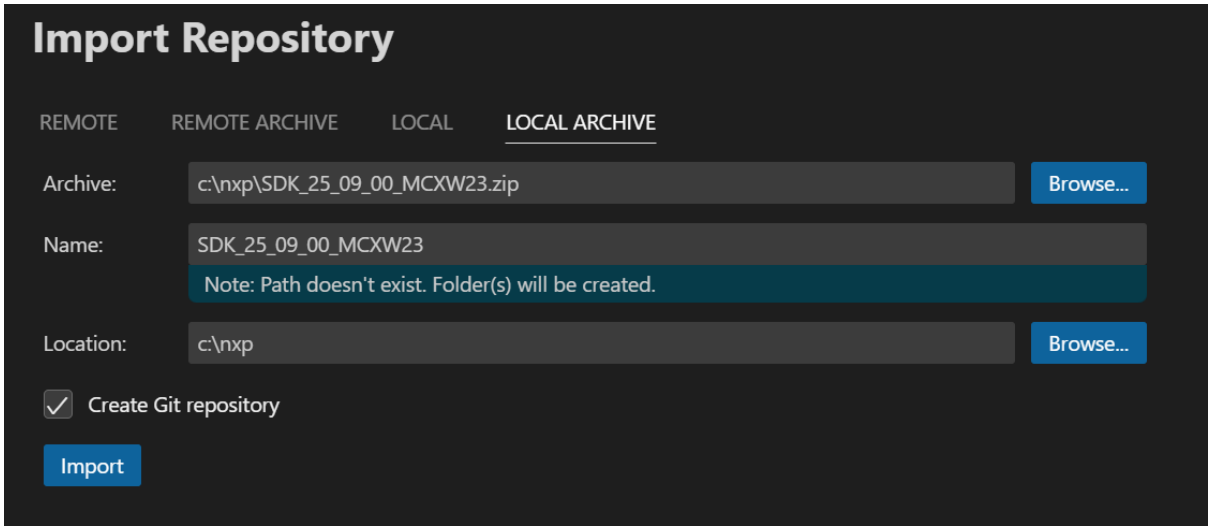
Import Repository-Layout SDK Package Click **Import Repository** from the **QUICKSTART PANEL**



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



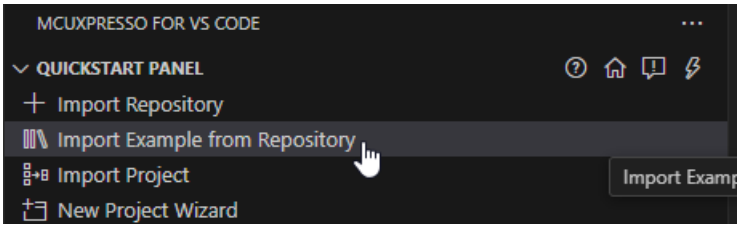
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



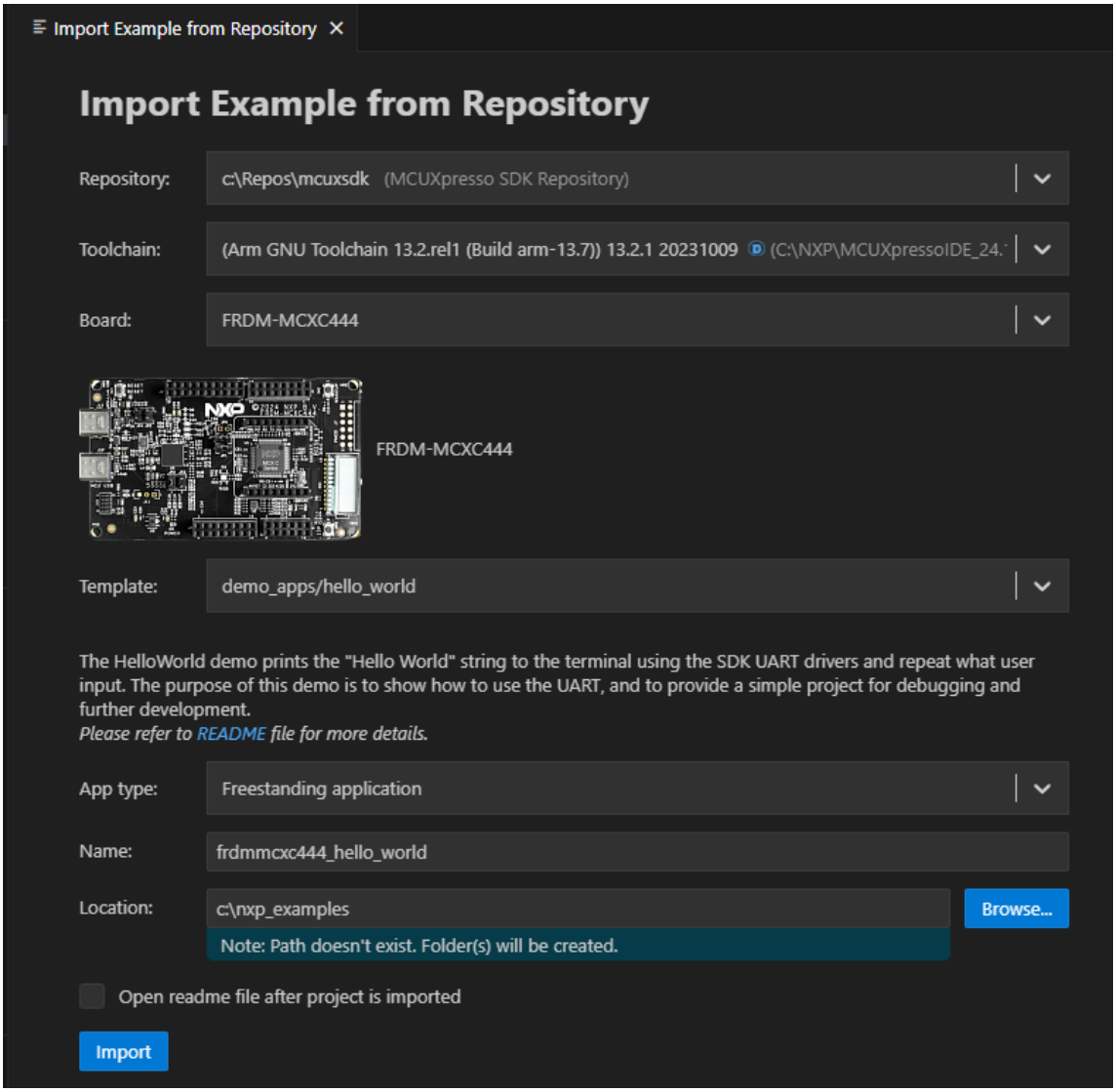
Building Example Applications

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:

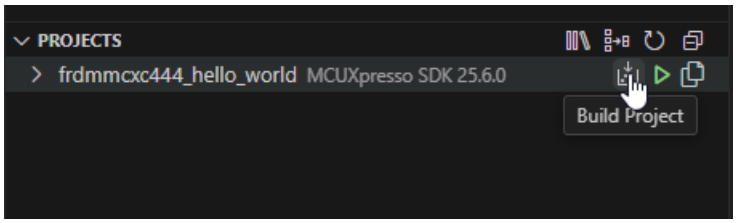
- Imported to user-defined location
- Independent of SDK location

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

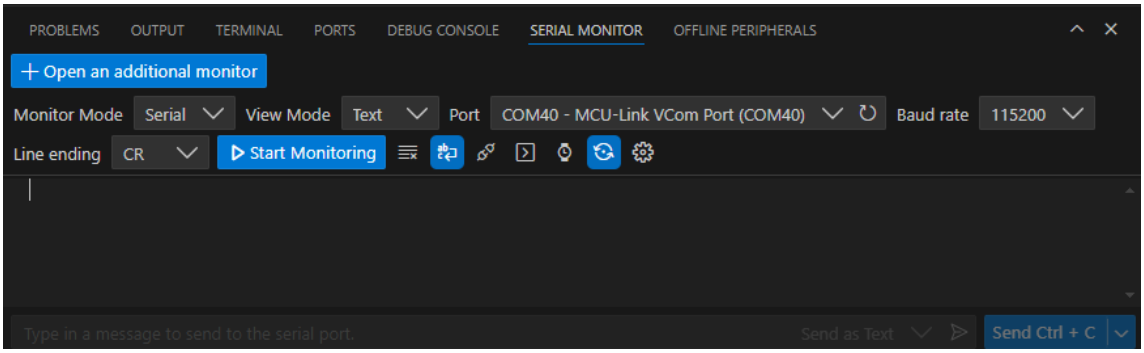


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

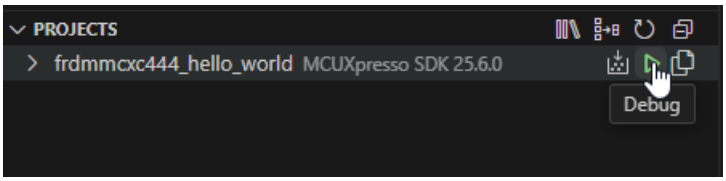
1. Open **Serial Monitor** from VS Code's integrated terminal



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

Debug Session

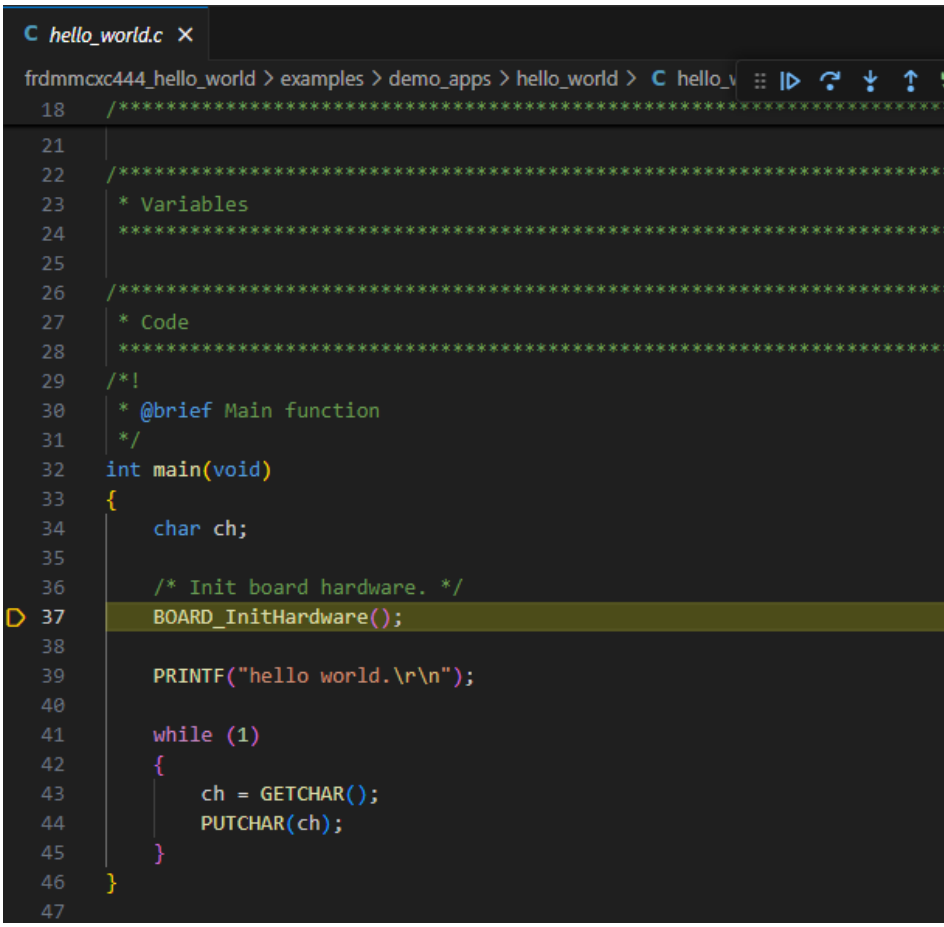
1. Navigate to **PROJECTS** view
2. Click the play button to initiate a debug session



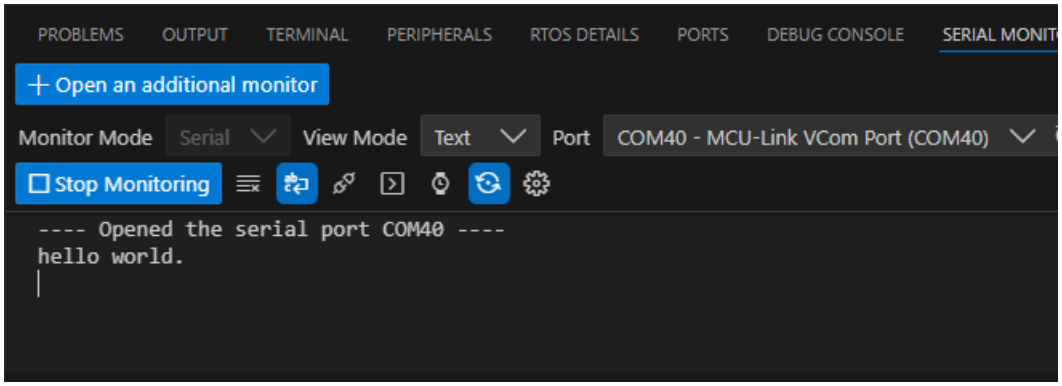
The debug session will begin with debug controls initially at the top of the interface.

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 Wiki DebugK section.

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 MCUXpresso for VS Code Wiki 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:

Directory	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 `west manifest config group.filter +optional` and `west update mcux-soc-svd` 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 frdmmcxw23

# 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., frdmmcxw23
west update_board --set board frdmmcxw23

# List available repositories for the board before updating
west update_board --set board frdmmcxw23 --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 frdmmcxw23 --list-repo

# Export repository information to YAML file for reference
west update_board --set board frdmmcxw23 --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 frdmmcxw23 -o frdmmcxw23-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 9.60.4
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

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

Development boards	MCU devices
FRDM-MCXE31B	**MCXE31BMPB**, MCXE315MLF, MCXE315MPA, MCXE316MLF, MCXE316MPA, MCXE317MPA, MCXE317MPB

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.

TF-M Trusted Firmware - M Library

PSA Test Suite Arm Platform Security Architecture Test Suite

IEC60730B Safety Library NXP IEC60730B Safety Library

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

mbedTLS mbedtls SSL/TLS library v3.x

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

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

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

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

ELE HSEB EdgeLock Secure Enclave (HSE B)

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

MCU Boot Open source MCU Bootloader.

LVGL LVGL Open Source Graphics Library

CANopenNode CANopenNode

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_nam
Utilities such as debug console	INSTALL_DIR/devices/<device_name>/utilities
Device Project Template for MCUXpresso IDE NPW	INSTALL_DIR/devices/<device_name>/project_templat
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

Board Support Files

board

[25.09.00]

- Updated BOARD_ClockPreConfig, switching MUX0 to safe clock before any configuration.

[25.09.00-pvw2]

- Added UTEST memory configuration in BOARD_ConfigMPU.

[25.09.00-pvw1]

- Updated boot_header supporting getting CM7 StartAddress from vector table.

[25.06.00]

- Initial version

clock_config

[25.06.00]

- Initial version

pin_mux

[25.09.00-pvw2]

- Added EMAC pins configuration.

[25.06.00]

- Initial version

BCTU

[2.1.0]

- New Feature
  - Add feature macro for compatibility with other SoCs.
- Improvements
  - Move _bctu_trig_mask and _bctu_trig_group from peripheral header to driver header.

[2.0.1]

- Improvements
  - Fixed CERT-C issues.

[2.0.0]

- Initial version.

CACHE ARMv7-M7

[2.0.5]

- Bug Fixes
  - Fixed cache operations to handle zero size and overflow in invalidate/clean functions

[2.0.4]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.3]

- Improvements
  - Deleted redundancy code about calculating cache clean/invalidate size and address aligns.

[2.0.2]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.1, 10.3 and 10.4.

[2.0.1]

- Bug Fixes
  - Fixed cache size issue in L2CACHE_GetDefaultConfig API.

[2.0.0]

- Initial version.

CLOCK

[2.2.0]

- Improvements
  - Added configurable timeout support for endless while loop.

[2.1.3]

- Bug fixes
  - Fixed the issue reported by CERT INT31-C checker.

[2.1.2]

- New features
  - Added EMAC_CLOCKS.

[2.1.1]

- Bug fix
  - Added missed mux source for MUX_9.
  - Delete some redundant mux source for MUX_10.

[2.1.0]

- New features
  - Added support for more devices.
  - Added FIRC divider support and API CLOCK_SetFircDiv.

[2.0.1]

- Bug fix
  - Fixed bugs in pll_unlock_accuracy_t and CLOCK_InitPll.
  - Fixed some potentioal operation wrap issue.

[2.0.0]

- initial version.

CMU_FC

[2.0.1]

- Bug Fixes
  - Fixed coverity issues

[2.0.0]

- Initial version.

CMU_FM

[2.0.1]

- Bug Fixes
  - Fixed coverity issues

[2.0.0]

- Initial version.

COMMON

[2.6.3]

- 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 irqs that mount under irqsteer interrupt extender.

[2.4.2]

- Improvements
  - Add the macros to convert peripheral address to secure address or non-secure address.

[2.4.1]

- Improvements
  - Improve for the macro redefinition error when integrated with zephyr.

[2.4.0]

- New Features
  - Added EnableIRQWithPriority, IRQ_SetPriority, and IRQ_ClearPendingIRQ for ARM.
  - Added MSDK_EnableCpuCycleCounter, MSDK_GetCpuCycleCount for ARM.

[2.3.3]

- New Features
  - Added NETC into status group.

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

- Bug Fixes
  - Fixed MAKE_VERSION overflow on 16-bit platforms.

[2.3.0]

- Improvements
  - Split the driver to common part and CPU architecture related part.

[2.2.10]

- Bug Fixes
  - Fixed the ATOMIC macros build error in cpp files.

[2.2.9]

- Bug Fixes
  - Fixed MISRA C-2012 issue, 5.6, 5.8, 8.4, 8.5, 8.6, 10.1, 10.4, 17.7, 21.3.
  - Fixed SDK_Malloc issue that not allocate memory with required size.

[2.2.8]

- Improvements
  - Included stddef.h header file for MDK tool chain.
- New Features:
  - Added atomic modification macros.

[2.2.7]

- Other Change
  - Added MECC status group definition.

[2.2.6]

- Other Change
  - Added more status group definition.
- Bug Fixes
  - Undef __VECTOR_TABLE to avoid duplicate definition in cmsis_clang.h

[2.2.5]

- Bug Fixes
  - Fixed MISRA C-2012 rule-15.5.

[2.2.4]

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

[2.2.3]

- New Features
  - Provided better accuracy of SDK_DelayAtLeastUs with DWT, use macro SDK_DELAY_USE_DWT to enable this feature.
  - Modified the Cortex-M7 delay count divisor based on latest tests on RT series boards, this setting lets result be closer to actual delay time.

[2.2.2]

- New Features
  - Added include RTE_Components.h for CMSIS pack RTE.

[2.2.1]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 3.1, 10.1, 10.3, 10.4, 11.6, 11.9.

[2.2.0]

- New Features
  - Moved SDK_DelayAtLeastUs function from clock driver to common driver.

[2.1.4]

- New Features
  - Added OTFAD into status group.

[2.1.3]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed the rule: rule-10.3.

[2.1.2]

- Improvements
  - Add SUPPRESS_FALL_THROUGH_WARNING() macro for the usage of suppressing fallthrough warning.

[2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

[2.1.0]

- New Features
  - Added IRQ operation for XCC toolchain.
  - Added group IDs for newly supported drivers.

[2.0.2]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed the rule: rule-10.4.

[2.0.1]

- Improvements
  - Removed the implementation of LPC8XX Enable/DisableDeepSleepIRQ() function.
  - Added new feature macro switch "FSL_FEATURE_HAS_NO_NONCACHEABLE_SECTION" for specific SoCs which have no noncacheable sections, that helps avoid an unnecessary complex in link file and the startup file.
  - Updated the align(x) to **attribute**(aligned(x)) to support MDK v6 armclang compiler.

[2.0.0]

- Initial version.

CRC

[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.

DCM

[2.0.1]

- Improvement
  - Support MCXE315, MCXE316.

[2.0.0]

- initial version.

DCM_GPR

[2.0.0]

- initial version.

DMAMUX

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EDMA

[2.10.9]

- 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.
  - Modfiy 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 scather/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 aovid 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 reduntant 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.

---

eMIOS

[2.0.0]

- Initial version.

ENET_QOS

[2.7.2]

- Bug Fixes
  - Not disabling ENET_QOS_ConfigureRxParser function when compiled for EMAC, as the MTL_RXP_INDIRECT_ACC_DATA register is writeable now.

[2.7.1]

- Bug Fixes
  - Fixed writing after DMA_CH array on platforms with smaller number of channels (EMAC).

[2.7.0]

- New features
  - The driver can work with EMAC IP now. EMAC is similar to ENET_QOS but doesn't support 100 % of its features.

[2.6.5]

- Bug Fixes
  - Fixed ENET_QOS_GetMacAddr address byte order not matching ENET_QOS_SetMacAddr.

[2.6.4]

- Improvements
  - ENET_QOS_SetMII returns success or failure status now (related to i.MX RT1170 errata ERR050539 - ENET_QOS doesn't support RMII 10Mbps mode).
- Bug Fixes
  - Fixed the MISRA C-2012 issue rule 14.3.

[2.6.3]

- Bug Fixes
  - Fixed the issue that ENET_QOS_GetRxFrame, ENET_QOS_ReadFrame and ENET_QOS_DropFrame did not properly restart the receiving once it stopped.

[2.6.2]

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

[2.6.1]

- Bug Fixes
  - Fixed the issue that ENET_QOS_ReadFrame doesn't check timestamp available bit before check the context BD bit, it makes software update extra BD. If DMA receives new frame to this BD before software update, software will lose this frame.

[2.6.0]

- New features
  - Added hardware checksum acceleration support.

[2.5.3]

- Bug Fixes
  - Fixed the MISRA issue rule 14.3, 5.3.

[2.5.2]

- Bug Fixes
  - Fixed the issue that ENET_QOS_Init reset the MDIO setting of ENET_QOS_SetSMI.

[2.5.1]

- Improvements
  - Supported RMII mode.

[2.5.0]

- Improvements
  - Added MDIO access wrapper APIs for ease of use.

[2.4.1]

- Improvements
  - Supported cache control.
  - Supported BD address convertion to system address.
  - Make driver aarch64 compatible
- Bug Fixes
  - Fixed the issue that driver internal interface ENET_QOS_DropFrame drops all frames in whole BD ring rather than one frame as design. Impact case: 1. Rx drop occurs in zero copy Rx API ENET_QOS_GetRxFrame. 2. Call ENET_QOS_ReadFrame with data pointer is NULL, driver will drop all Rx frames.

[2.4.0]

- New features
  - Added MDIO IEEE802.3 Clause 45 access support.
  - Added get statistics API to get some statistical data in transfer.
  - Added new APIs to support zero copy Rx.
  - Fixed the MISRA issue rule 8.4, 8.6.

[2.3.0]

- Improvements
  - Added counter to record multicast hash conflict in struct _enet_handle, improved the situation that one multicast group could be left by other conflict multicast address left operation.
- Bug Fixes
  - Updated txDirtyRing maintenance in reclaim and send frame process, allow txDirtyRing to be overwritten.
  - Disabled carrier sensing in full duplex mode configuration in ethernet initialization
  - Fixed 1588 sub-second calculate issue.

[2.2.2]

- Bug Fixes
  - Fixed the issue that ENET_QOS_SetupTxDescriptor didn't handle the DMA access address mapping for SoCs have feature FSL_FEATURE_MEMORY_HAS_ADDRESS_OFFSET.
  - Fixed MISRA 2012 violations detected in examples build.

[2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 violations, fixed doxygen warning.
  - Fixed the issue that cache invalidate to invalid converted memory address in ENET_QOS_ReadFrame for SoCs have feature FSL_FEATURE_MEMORY_HAS_ADDRESS_OFFSET.

[2.2.0]

- Removed the ptp time data ring management, below structures and APIs are removed:
  - structure enet_qos_ptp_time_data_t
  - structure enet_qos_ptp_time_data_ring_t
  - API ENET_QOS_GetRxFrameTime
  - API ENET_QOS_GetTxFrameTime
- Added API for GCL list read and AVB configuration
  - ENET_QOS_EstReadGcl
  - ENET_QOS_AVBConfigure

- Improved driver for PTP system time configuration, timestamp read.
- Added IRQ lock and memory barrier instruction for descriptor operation.
- Fixed MISRA 2012 violations

[2.1.1]

- Bug Fixes
  - Fixed the bug that data pointer is not converted to local memory address in the call to ENET_QOS_Ptp1588ParseFrame.

[2.1.0]

- New feature
  - Update driver to support feature FSL_FEATURE_MEMORY_HAS_ADDRESS_OFFSET which convert buffer address to visible address for DMA.
  - Require user to provide implementation for ENET_QOS_SetSYSControl API, which set the PHY interface and enable clock generation for IP.

[2.0.0]

- Initial version.

FLEXCAN

[2.14.5]

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

[2.14.4]

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

[2.14.3]

- Improvements
  - Add unhandled interrupt events check for following API:
    - FLEXCAN_MbHandleIRQ
    - FLEXCAN_EhancedRxFifoHandleIRQ
- Bug Fixes

  - Remove FLEXCAN_MemoryErrorHandleIRQ on some platform without memory error interrupt.
  - Add conditional compile for CTRL2[ISOCANFDEN] because some platform do not have this bit.

[2.14.2]

- Improvements
  - Add Coverage Justification for uncovered code.
  - Adjust API FLEXCAN_TransferAbortReceive order.
  - Update FLEXCAN_Enable to enter Freeze Mode first when enter Disable mode on some platform.
  - Added while loop timeout for following API:
    * FLEXCAN_EnterFreezeMode
    * FLEXCAN_ExitFreezeMode
    * FLEXCAN_Enable
    * FLEXCAN_Reset
    * FLEXCAN_TransferSendBlocking
    * FLEXCAN_TransferReceiveBlocking
    * FLEXCAN_TransferFDSendBlocking
    * FLEXCAN_TransferFDReceiveBlocking
    * FLEXCAN_TransferReceiveFifoBlocking
    * FLEXCAN_TransferReceiveEnhancedFifoBlocking
- Bug Fixes
  - Remove remote frame feature in CANFD mode because there is no remote frame in the CANFD format.
  - Remove legacy Rx FIFO disabled branch in FLEXCAN_SubHandlerForLegacyRxFIFO and FLEXCAN_SubHandlerForDataTransfered.

[2.14.1]

- Bug Fixes
  - Fixed register IMASK2-4 IFLAG2-4 HR_TIME_STAMPn access issue on FlexCAN instances with different number of MBs.
  - Fixed bit field MBDSR1-3 access issue on FlexCAN instances with different number of MBs.
- Improvements
  - Unified following API as same parameter and return value type:
    * FLEXCAN_GetMbStatusFlags
    * FLEXCAN_ClearMbStatusFlags
    * FLEXCAN_EnableMbInterrupts
    * FLEXCAN_DisableMbInterrupts
  - Add workaround for ERR050443 and ERR052403.

  - Update message buffer read process in API FLEXCAN_ReadRxMb and FLEXCAN_ReadFDRxMb to make critical section as short as possible.
  - Simplify API FLEXCAN_DriverDataIRQHandler implementation by remove parameter type.

[2.14.0]

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

[2.13.1]

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

[2.13.0]

- Improvements
  - Support payload endianness selection feature.

[2.12.0]

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

[2.11.8]

- Improvements
  - Synchronize flexcan driver update on s32z platform.

[2.11.7]

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

[2.11.6]

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

[2.11.5]

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

[2.11.4]

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

[2.11.3]

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

[2.11.2]

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

[2.11.1]

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

[2.11.0]

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

[2.10.1]

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

[2.10.0]

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

[2.9.2]

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

[2.9.1]

- Bug Fixes
  - Fixed the issue that FLEXCAN_TransferReceiveEnhancedFifoBlocking() API clearing Fifo data available flag more than once.
  - Fixed the issue that entering FLEXCAN_SubHandlerForEhancedRxFifo() even if Enhanced Rx fifo interrupts are not enabled.
  - Fixed the issue that FLEXCAN_TransferReceiveEnhancedFifoEDMA() update handle even if previous Rx FIFO receive not finished.

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

[2.9.0]

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

[2.8.7]

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

[2.8.6]

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

[2.8.5]

- Improvements
  - Make driver aarch64 compatible.

[2.8.4]

- Bug Fixes
  - Fixed FlexCan_Errata_6032 to disable all interrupts.

[2.8.3]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.2]

- Improvements
  - Add CANFD frame data length enumeration.

[2.6.1]

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

[2.6.0]

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

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Bug Fixes
  - MISRA C-2012 issue check.
    - Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-11.8, rule-12.2, rule-13.4, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-16.4, rule-17.3, rule-5.8, rule-8.3, rule-8.5.
  - Fixed the issue that API FLEXCAN_SetFDRxMbConfig lacks inactive message buff.
  - Fixed the issue of Pa082 warning.
  - Fixed the issue of dead lock in the function of interruption handler.
  - Fixed the issue of Legacy Rx Fifo EDMA transfer data fail in evkmimxrt1060 and evkmimxrt1064.
  - Fixed the issue of setting CANFD Bit Rate Switch.
  - Fixed the issue of operating unknown pointer risk.
    - when used the pointer "handle->mbFrameBuf[mbIdx]" to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
  - Added assert to check current CAN clock source affected by other clock gates in current device.
    - In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).
- Improvements
  - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
  - Optimized the function of interruption handler.
  - Added two APIs for FLEXCAN EDMA driver.
    - FLEXCAN_PrepareTransfConfiguration
    - FLEXCAN_StartTransferDatafromRxFIFO
  - Added new API for FLEXCAN driver.
    - FLEXCAN_GetTimeStamp
      - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
    - FLEXCAN_EnterFreezeMode

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

- Improvements
  - Hid FLEXCAN_Reset() from user.
  - Used NDEBUG macro to wrap FLEXCAN_IsMbOccupied() function instead of DEBUG macro.

[2.0.0]

- Initial version.

FLEXCAN_EDMA

[2.12.1]

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

[2.12.0]

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

[2.11.7]

- Refer FLEXCAN driver change log 2.7.0 to 2.11.7

FLEXIO

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
  - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

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

[2.1.0]

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

[2.0.4]

- Bug Fixes
  - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.2]

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

[2.0.1]

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

FLEXIO_I2C

[2.6.2]

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

[2.6.1]

- Bug Fixes
  - Fixed coverity issues

[2.6.0]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.0]

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

[2.2.0]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

FLEXIO_I2S_EDMA

[2.1.9]

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

[2.1.8]

- Improvements
  - Applied EDMA ERRATA 51327.

FLEXIO_MCU_LCD

[2.3.0]

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

[2.2.0]

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

[2.1.0]

- New Features
  - Supported transmit only data without command.

[2.0.8]

- Bug Fixes
  - Fixed bug that FLEXIO_MCULCD_Init return kStatus_Success even with invalid parameter.
  - Fixed glitch on WR, that when initially configure the timer pin as output, or change the pin back to disabled, the pin may be driven low causing glitch on bus. Configure the pin as bidirection output first then perform a subsequent write to change to output or dsiabled 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 Dozen mode configuration error in FLEXIO_UART_Init API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
  - Added code to report errors if the user sets a too-low-baudrate which FLEXIO cannot reach.
  - Disabled FLEXIO_UART receive interrupt instead of all NVICs when reading data from ring buffer. If ring buffer is used, receive nonblocking will disable all NVIC interrupts to protect the ring buffer. This had negative effects on other IPs using interrupt.

[2.1.1]

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

[2.1.0]

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

FLEXIO_UART_EDMA

[2.3.1]

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

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0

INTM

[2.1.0]

- Replace macro FSL_FEATURE_INTM_MONITOR_COUNT to INTM_MON_COUNT.

[2.0.1]

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

[2.0.0]

- Initial version.

LCU

[2.0.0]

- Initial version.

LPCMP

[2.3.2]

- Improvements
  - Fixed LPCMP CERT-C issues.

[2.3.1]

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

[2.3.0]

- New Feature
  - Added some new features for platforms which support
    - Plus input source selection.
    - Minus input source selection.
    - CMP to DAC link.
- Improvements

  - Removed some new features for platforms which doesn't support
    * Functional clock source selection.
    * DAC high power mode selection.
    * Round Robin clock source selection.
    * Round Robin trigger source selection.
    * Round Robin channel sample numbers setting.
    * Round Robin channel sample time threshold setting.
    * Round Robin internal trigger configuration.

[2.2.0]

- Improvements
  - Change FSL_FEATURE_LPCMP_HAS_NO_CCR0_CMP_STOP_EN to FSL_FEATURE_LPCMP_HAS_CCR0_CMP_STOP_EN.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPI2C

[2.6.3]

- 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^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 unsuccesful 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

[2.4.6]

- 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_SetPCSContinous when disabling PCS continous 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 continous 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_SetPCSContinous to support changing PCS selection and PCS continous mode.
- Bug Fixes
  - Fixed bug in non-blocking and EDMA transfer APIs that kStatus_InvalidArgument is returned if user configures 3-wire mode and full-duplex transfer at the same time, but transfer state is already set to kLPSPI_Busy by mistake causing following transfer can not start.
  - Fixed bug when LPSPI slave using EDMA way to transfer, tx should be masked when tx data is null, otherwise in 3-wire mode which tx/rx use the same pin, the received data will be interfered.

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPSPI_EDMA

[2.4.9]

- Improvements
  - Removed unused code from LPSPI_SeparateEdmaReadData().

[2.4.8]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

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

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

- Improvements
  - Add the TXMSK wait after TCR setting.

[2.4.0]

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

LPUART

[2.10.0]

- New Feature
  - Added support to configure RTS watermark.

[2.9.4]

- Improvements
  - Merged duplicate code.

[2.9.3]

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

[2.9.2]

- Bug Fixes
  - Fixed coverity issues.

[2.9.1]

- Bug Fixes
  - Fixed coverity issues.

[2.9.0]

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

[2.8.3]

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

[2.8.2]

- Bug Fix
  - Fixed the bug that LPUART_TransferEnable16Bit 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 lupart 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 acurate baudrate setting. Changed osr from uint32_t to uint8_t since osr's bigest 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

MC_RGM

[2.1.0]

- New Features
  - Added MC_RGM_DisableBidirectionalReset to make external reset pin not asserted on a given 'functional' reset event.
- Improvements
  - Remove redundant enum types.

[2.0.0]

- Initial version.

MCM

[2.2.0]

- Improvements
  - Support platforms with less features.

[2.1.0]

- Others
  - Remove byteID from mcm_lmem_fault_attribute_t for document update.

[2.0.0]

- Initial version.

MSCM

[2.0.0]

- Initial version.

PIT

[2.2.0]

- Bug Fixes
  - According to ERR050763, PIT_LDVAL_STAT register is not reliable in dynamic load mode, so remove the status check in PIT_SetRtiTimerPeriod which added since 2.1.1.
  - Removed not used bit PIT_RTI_TCTRL_CHN_MASK.
- Improvements
  - Added more guide about get RTI load status in PIT_SetRtiTimerPeriod's API comment.
  - Change PIT_RTI_Deinit to inline API.
  - Ensure PIT peripheral clock enabled in PIT_RTI_Init.
- New Features
  - Added PIT_ClearRtiSyncStatus API to clear the RTI_LDVAL_STAT register.

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

POWER

[2.0.0]

- initial version.

QSPI

[2.3.1]

- Improvements
  - Fixed Coverity MSG issues.

[2.3.0]

- New Features
  - Applied the QSPI IP update with register field changes.
  - Added Soc specific driver to integrate Soc configuration.
- Changed
  - Updated the QSPI LUT update function to be compatible with different sequence unit.
  - Added new feature macro FSL_FEATURE_QSPI_HAS_SOC_SPECIFIC_CONFIG which represents there're Soc specific QSPI configurations. Soc specific driver should cover these configurations. Previous Soc specific code in the common driver should be masked.

[2.2.5]

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

[2.2.4]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3.

[2.2.3]

- Bug Fixes
  - Cleared buffer generic configuration when do software reset.

[2.2.2]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1 and 11.9.

[2.2.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.6, 10.8, 11.3, 11.6, 11.8, 11.9, 14.4, 16.1, 16.4, 17.7.

[2.2.0]

- New Features
  - Added new API QSPI_ClearCache to clear cache for new IP feature FSL_FEATURE_QSPI_SOCCR_HAS_CLR_LPCAC.
- Bug Fixes
  - Fixed the QSPI_WriteBlocking API programming issue for low watermark, caused by previous improvement change of using TX watermark signal to fill the TX FIFO. Reverted change to previous implementation to use TX FIFO full flag for filling the FIFO. Improved previous API by accessing TX data register directly.
  - Fixed the issue that QSPI_SetIPCommandSize incorrectly triggered a transaction.
  - Fixed clock divider accurate issue when using internal QSPI internal divider.
  - Fixed build fail issue for some devices' not supporting API QSPI_SetDqsConfig for DQS configuration.

[2.1.0]

- New Features
  - Added new API QSPI_SetDqsConfig for DQS configuration.
- Improvements
  - Updated the QSPI_WriteBlocking API to fill the TX FIFO once there are bytes of TX watermark room in the FIFO. This will improve the performance of filling TX FIFO when watermark is high.

[2.0.2]

- Improvements
  - New Macro function:
    * Added QSPI_LUT_SEQ() function for users to set LUT table easily.
    * Added LUT command macros for users to easy use.
  - Comment update:
    * Added the comments for the limitation of QSPI_ReadBlocking and QSPI_TransferReceiveBlocking.

[2.0.1]

- Improvements
  - New API:
    * QSPI_SetReadArea to set the read area.
- Bug Fixes
  - Fixed the issue that QSPI_UpdateLUT function only updated first LUT.
  - Fixed issue that some function that hardcode QSPI0 as base.

[2.0.0]

- Initial version.

QSPI_EDMA

[2.2.4]

- Changed
  - Compatible with new EDMA driver.

[2.2.3]

- Changed
  - Used instance array number to count the QSPI instead of feature macro.

[2.0.0]

- Initial version.

RTC

[2.0.0]

- Initial version.

SAI

[2.4.10]

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

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

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

[2.4.7]

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

[2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

[2.4.5]

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

[2.4.4]

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

[2.4.3]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
  - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

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

[2.3.6]

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

[2.3.5]

- Improvements
  - Make driver to be aarch64 compatible.

[2.3.4]

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

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.3]

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

[2.2.2]

- Bug Fixes
  - Fixed the issue of MISRA 2004 rule 9.3.
  - Fixed sign-compare warning.
  - Fixed the PA082 build warning.
  - Fixed sign-compare warning.
  - Fixed violations of MISRA C-2012 rule 10.3,17.7,10.4,8.4,10.7,10.8,14.4,17.7,11.6,10.1,10.6,8.4,14.3,16.4,18
  - Allow to reset Rx or Tx FIFO pointers only when Rx or Tx is disabled.
- Improvements
  - Added 24bit raw audio data width support in sai sdma driver.
  - Disabled the interrupt/DMA request in the SAI_Init to avoid generates unexpected sai FIFO requests.

[2.2.1]

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

[2.2.0]

- Improvements
  - Added new APIs for parameters collection and simplified user interfaces:
    - SAI_Init
    - SAI_SetMasterClockConfig
    - SAI_TxSetBitClockRate
    - SAI_TxSetSerialDataConfig
    - SAI_TxSetFrameSyncConfig
    - SAI_TxSetFifoConfig
    - SAI_TxSetBitclockConfig
    - SAI_TxSetConfig
    - SAI_TxSetTransferConfig
    - SAI_RxSetBitClockRate
    - SAI_RxSetSerialDataConfig
    - SAI_RxSetFrameSyncConfig
    - SAI_RxSetFifoConfig
    - SAI_RxSetBitclockConfig
    - SAI_RXSetConfig
    - SAI_RxSetTransferConfig
    - SAI_GetClassicI2SConfig
    - SAI_GetLeftJustifiedConfig
    - SAI_GetRightJustifiedConfig
    - SAI_GetTDMConfig

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - API name changes:
    - SAI_GetSendRemainingBytes -> SAI_GetSentCount.
    - SAI_GetReceiveRemainingBytes -> SAI_GetReceivedCount.
    - All names of transactional APIs were added with “Transfer” prefix.
    - All transactional APIs use base and handle as input parameter.
    - Unified the parameter names.
- Bug Fixes
  - Fixed WLC bug while reading TCSR/RCSR registers.
  - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI_TxInit/SAI_RxInit.

[2.0.0]

- Initial version.

SAI_EDMA

[2.7.3]

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

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
  - Removed deprecated APIs.

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2

SAR_ADC

[2.3.0]

- New Feature
  - Added new feature macro a for compatibility with ADCs on some platforms where some instances do not support group3.

[2.2.0]

- New Feature
  - Added new features to compatible with new platforms.

[2.1.1]

- Improvement
  - Change ADC sample rate phase duration default value from 0x08 to 0x14.

[2.1.0]

- New Feature
  - Added ADC_StopConvChain function to support stop scan in normal conversion scan operation mode.

[2.0.3]

- Bug Fixes
  - Fixed the array name usage error in function ADC_GetInstance.

[2.0.2]

- Bug Fixes
  - Fixed MISRA issues.

[2.0.1]

- Bug Fixes
  - Fixed the bug that when calling function ADC_EnableWdgThresholdInt() in function ADC_SetAnalogWdgConfig(), the parameter was passed incorrectly.

[2.0.0]

- Initial version.

SEMA42

[2.1.1]

- Improvements
  - Updated SEMA42_TryLock function to avoid unsigned integer operations wrap issue.

[2.1.0]

- New Features
  - Added SEMA42_BUSY_POLL_COUNT parameter to prevent infinite polling loops in SEMA42 operations.
  - Added timeout mechanism to all polling loops in SEMA42 driver code.
- Improvements
  - Updated SEMA42_Lock function to return status_t instead of void for better error handling.
  - Enhanced documentation to clarify timeout behavior and return values.

[2.0.4]

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

[2.0.3]

- Improvements
  - Changed to implement SEMA42_Lock base on SEMA42_TryLock.

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

SIUL2

[2.1.0]

- New features
  - Added EIRQ related APIs EIRQ SIUL2_GetPinEirqNumber, SIUL2_EnableExtInteruptWithCallback, SIUL2_DisableExtInterruptCallback.

[2.0.3]

- New features
  - Updated siul2_adc_interleaves_t to support MCXE315, MCXE316, MCXE317 parts.

[2.0.2]

- New features
  - Updated SIUL2_SetPinInputBuffer to handle kPORT_INPUT_MUX_NO_INIT.

[2.0.1]

- Bug fix
  - Fixed some potentioal operation wrap issue.

[2.0.0]

- initial version.

STM

[2.0.1]

- Bug Fixes
  - Fixed coverity and msg violations caused by potential out of bounds array operation.

[2.0.0]

- Initial version.

SWT

[2.1.1]

- Improvements
  - Handle errata ERR052226: Toggling watchdog enable may cause unexpected timeout in some boundary conditions. Update SWT_SetTimeoutValue() update SWT with the watchdog keys to restart the counter value before setting a new timeout value.

[2.1.0]

- Bug Fixes
  - Fixed CERT-C violations: CERT INT31-C, CERT INT30-C
- New Features
  - Added SWT_Refresh() API to automatically select a correct refresh service mode.

[2.0.0]

- Initial version.

TEMPSENSE

[2.0.0]

- Initial version.

TRGMUX

[2.0.1]

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

[2.0.0]

- Initial version.

TSPC

[2.0.0]

- Initial version.

VIRT_WRAPPER

[2.0.0]

- Initial version.

WKPU

[2.0.0]

- Initial version.

XBIC

[2.0.1]

- Bug Fixes
  - Add __DSB() to avoid instruction optimization issue.

[2.0.0]

- Initial version.

XRDC

[2.0.7]

- Improvements
  - Handle errata ERR050593.

[2.0.6]

- Improvements
  - Supported platforms which don't have XRDC clock gate control.

[2.0.5]

- Bug Fixes
  - Fixed XRDC_GetAndClearFirstSpecificDomainError potential array over index issue.

[2.0.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 3.1, 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 11.3, 12.2, 14.4, 17.7, 20.7.

[2.0.3]

- Improvements
  - Added necessary driver supports for K32H844P.
  - Added new APIs concerning new features of Exclusive Access Lock and programmable domain access flags configurations.

[2.0.2]

- Bug Fixes
  - Fixed wrong assert of assignIndex input check in the xRDC driver.
- Improvements
  - Added master input CPU/non-CPU check in XRDC_SetNonProcessorDomainAssignment and XRDC_SetProcessorDomainAssignment API.
  - Added necessary assert checks for several config inputs.

[2.0.1]

- Improvements
  - Changed reserved bit fields in the structs into unnamed-identifier bit fields.

[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.

MCXE31B

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 FreeRTOS

FreeRTOS

Chapter 2

MCXE31B

2.1 BCTU: BCTU Module

void BCTU_GetDefaultConfig(*bctu_config_t* *config)

Gets the default configuration for BCTU.

Parameters

- config – Pointer to BCTU configuration structure.

void BCTU_Init(BCTU_Type *base, const *bctu_config_t* *config)

Initializes the BCTU.

Parameters

- base – BCTU peripheral base address.
- config – Pointer to BCTU configuration structure.

void BCTU_Deinit(BCTU_Type *base)

Deinitializes the BCTU.

Parameters

- base – BCTU peripheral base address.

void BCTU_SetConvListConfig(BCTU_Type *base, const *bctu_convlist_config_t* *config, uint8_t convListIndex)

BCTU conversion list configuration.

Parameters

- base – BCTU peripheral base address.
- config – Pointer to BCTU conversion list configuration structure.
- convListIndex – conversion list index.

static inline void BCTU_AssertSoftwareReset(BCTU_Type *base, bool enable)

Assert/Deassert BCTU software reset.

Parameters

- base – BCTU peripheral base address.
- enable – Indicates whether to assert or deassert BCTU software reset.
  - **true** Assert BCTU software reset.
  - **false** Deassert BCTU software reset.

static inline void BCTU_EnableModule(BCTU_Type *base, bool enable)

Enable/Disable BCTU.

Parameters

- base – BCTU peripheral base address.
- enable – Indicates whether to enable or disable the BCTU module.
  - **true** Enable BCTU module.
  - **false** Disable BCTU module.

static inline void BCTU_EnableDmaTrans(BCTU_Type *base, enum *_bctu_trig_adc* instance, bool enable)

Enable/Disable DMA operation when new data is available in data result register.

Parameters

- base – BCTU peripheral base address.
- instance – ADC instance _bctu_trig_adc.
- enable – Indicates whether to enable or disable the DMA operation.
  - **true** Enable the DMA operation.
  - **false** Disable the DMA operation.

static inline void BCTU_EnableInt(BCTU_Type *base, uint32_t mask, bool enable)

Enable/Disable BCTU interrupts.

Parameters

- base – BCTU peripheral base address.
- mask – BCTU interrupt mask, _bctu_int.
- enable – Indicates whether to enable or disable the data interrupt.
  - **true** Enable interrupt.
  - **false** Disable interrupt.

static inline uint32_t BCTU_GetStatusFlags(BCTU_Type *base)

Get BCTU status flags.

Parameters

- base – BCTU peripheral base address._bctu_status_flags.

Returns

BCTU status flags, _bctu_status_flags.

static inline void BCTU_ClearStatusFlags(BCTU_Type *base, uint32_t mask)

Clear BCTU status flags.

Parameters

- base – BCTU peripheral base address.
- mask – Mask value for flags to be cleared, refer _bctu_status_flags.

void BCTU_SetTrigConfig(BCTU_Type *base, const *bctu_trig_config_t* *config)

BCTU trigger configuration.

Parameters

- base – BCTU peripheral base address.
- config – Pointer to BCTU trigger configuration structure.

static inline void BCTU_EnableGlobalTrig(BCTU_Type *base, bool enable)

Enable/Disable global trigger.

Parameters

- base – BCTU peripheral base address.
- enable – Indicates whether to enable or disable the global trigger.
  - **true** Enable global trigger.
  - **false** Disable global trigger.

static inline void BCTU_EnableHardwareTrig(BCTU_Type *base, enum _bctu_trig_source trigIndex, bool enable)

Enable BCTU hardware trigger.

Parameters

- base – BCTU peripheral base address.
- trigIndex – Trigger index, _bctu_trig_source
- enable – Indicates whether to enable or disable the hardware trigger.
  - **true** Enable hardware trigger.
  - **false** Disable hardware trigger.

static inline void BCTU_EnableSoftwareTrig(BCTU_Type *base, *bctu_trig_group_t* trigGroup, uint32_t trigMask)

Enable BCTU software trigger.

Parameters

- base – BCTU peripheral base address.
- trigGroup – Trigger group, bctu_trig_group_t
- trigMask – Trigger mask, _bctu_trig_mask

static inline void BCTU_GetFifoResult(BCTU_Type *base, *bctu_fifo_t* fifoIndex, *bctu_fifo_res_t* *result)

Get BCTU FIFO result data.

Parameters

- base – BCTU peripheral base address.
- fifoIndex – FIFO index bctu_fifo_t.
- result – Structure to obtain BCTU FIFO result, bctu_fifo_res_t.

static inline void BCTU_SetFifoWaterMark(BCTU_Type *base, *bctu_fifo_t* fifoIndex, uint8_t watermark)

Set BCTU FIFO watermark.

Parameters

- base – BCTU peripheral base address.
- fifoIndex – FIfo index bctu_fifo_t.
- watermark – FIFO watermark.

static inline void BCTU_EnableFifoDma(BCTU_Type *base, *bctu_fifo_t* fifoIndex, bool enable)

Enable/Disable FIFO DMA transfer.

Parameters

- base – BCTU peripheral base address.

- fifoIndex – FIFO index bctu_fifo_t.
- enable – Indicates whether to enable or disable the FIFO DMA transfer.
  - **true** Enable the FIFO DMA transfer.
  - **false** Disable the FIFO DMA transfer.

static inline void BCTU_EnableFifoInt(BCTU_Type *base, uint32_t mask, bool enable)

Enable/Disable FIFO interrupt.

Parameters

- base – BCTU peripheral base address.
- mask – BCTU FIFO interrupt mask, _bctu_fifo_int.
- enable – Indicates whether to enable or disable the FIFO interrupt.
  - **true** Enable the FIFO interrupt.
  - **false** Disable the FIFO interrupt.

static inline uint32_t BCTU_GetFifoStatusFlags(BCTU_Type *base)

Get BCTU FIFO status flag.

Parameters

- base – BCTU peripheral base address.

Returns

FIFO status flags, _bctu_fifo_status_flags.

static inline void BCTU_ClearFifoStatusFlags(BCTU_Type *base, uint32_t mask)

Clear BCTU status flags.

Parameters

- base – BCTU peripheral base address.
- mask – Mask value for flags to be cleared, refer _bctu_fifo_status_flags.

static inline bool BCTU_GetFifoFullFlag(BCTU_Type *base, *bctu_fifo_t* fifoIndex)

Get BCTU FIFO full flag.

Parameters

- base – BCTU peripheral base address.
- fifoIndex – FIFO index bctu_fifo_t.

Returns

FIFO full flag.

- **true** FIFO full.
- **false** FIFO not full.

static inline uint8_t BCTU_GetFifoCounter(BCTU_Type *base, *bctu_fifo_t* fifoIndex)

Get BCTU FIFO counter.

Parameters

- base – BCTU peripheral base address.
- fifoIndex – FIFO index bctu_fifo_t.

Returns

FIFO counter.

static inline void BCTU_GetConvResult(BCTU_Type *base, enum *_bctu_trig_adc* instance, *bctu_adc_conv_result_t* *result)

Get ADC conversion result.

Parameters

- base – BCTU peripheral base address.
- instance – ADC instance _bctu_trig_adc.
- result – Structure to obtain ADC conversion result, bctu_adc_conv_result_t.

FSL_BCTU_DRIVER_VERSION

BCTU driver version 2.1.0.

enum _bctu_int

BCTU interrupt mask.

Values:

enumerator kBCTU_NewDataInt_0

Enables an interrupt request when BCTU writes a new conversion result to ADC data register 0.

enumerator kBCTU_NewDataInt_1

Enables an interrupt request when BCTU writes a new conversion result to ADC data register 1.

enumerator kBCTU_ConvListInt

Enables an interrupt request for the last conversion in a conversion list.

enumerator kBCTU_TrigIn

Enables an interrupt request on a trigger flag.

enum _bctu_status_flags

BCTU status flags.

Values:

enumerator kBCTU_NewData_0_Ready

Indicates that new conversion data is available in ADC data register 0.

enumerator kBCTU_NewData_1_Ready

Indicates that new conversion data is available in ADC data register 1.

enumerator kBCTU_NewData_0_OverWrite

Indicates that the data in ADC data register 0 has been overwritten with new data.

enumerator kBCTU_NewData_1_OverWrite

Indicates that the data in ADC data register 1 has been overwritten with new data.

enumerator kBCTU_ConvList_0_LastConvExecuted

Indicates that ADC0 has executed the last conversion in a conversion list.

enumerator kBCTU_ConvList_1_LastConvExecuted

Indicates that ADC1 has executed the last conversion in a conversion list.

enumerator kBCTU_Trig

Indicates at least one ADC was triggered.

enum _bctu_fifo_int

BCTU FIFO Interrupt mask.

Values:

enumerator kBCTU_FIfo_1_Int
Enables FIFO1 interrupt.

enumerator kBCTU_FIfo_2_Int
Enables FIFO2 interrupt.

enum _bctu_fifo_status_flags
BCTU FIFO status flags.

Values:

enumerator kBCTU_Fifo_1_Int
Indicates the number of active entries in FIFO1 exceeds the watermark level.

enumerator kBCTU_Fifo_2_Int
Indicates the number of active entries in FIFO2 exceeds the watermark level.

enumerator kBCTU_Fifo_1_OverRun
Indicates you have attempted to write to full FIFO1.

enumerator kBCTU_Fifo_2_OverRun
Indicates you have attempted to write to full FIFO2.

enumerator kBCTU_Fifo_1_UnderRun
Indicates you have attempted to read from empty FIFO1.

enumerator kBCTU_Fifo_2_UnderRun
Indicates you have attempted to read from empty FIFO2.

enum _bctu_trig_adc
BCTU trigger ADCn to convert.

Values:

enumerator kBCTU_TrigAdc_0
Trigger ADC 0 to convert.

enumerator kBCTU_TrigAdc_1
Trigger ADC 1 to convert.

enum _bctu_trig_mask
BCTU software trigger mask.

Values:

enumerator kBCTU_TrigMask_0
Trigger mask 0.

enumerator kBCTU_TrigMask_1
Trigger mask 1.

enumerator kBCTU_TrigMask_2
Trigger mask 2.

enumerator kBCTU_TrigMask_3
Trigger mask 3.

enumerator kBCTU_TrigMask_4
Trigger mask 4.

enumerator kBCTU_TrigMask_5
Trigger mask 5.

enumerator kBCTU_TrigMask_6
Trigger mask 6.

enumerator kBCTU_TrigMask_7
Trigger mask 7.

enumerator kBCTU_TrigMask_8
Trigger mask 8.

enumerator kBCTU_TrigMask_9
Trigger mask 9.

enumerator kBCTU_TrigMask_10
Trigger mask 10.

enumerator kBCTU_TrigMask_11
Trigger mask 11.

enumerator kBCTU_TrigMask_12
Trigger mask 12.

enumerator kBCTU_TrigMask_13
Trigger mask 13.

enumerator kBCTU_TrigMask_14
Trigger mask 14.

enumerator kBCTU_TrigMask_15
Trigger mask 15.

enumerator kBCTU_TrigMask_16
Trigger mask 16.

enumerator kBCTU_TrigMask_17
Trigger mask 17.

enumerator kBCTU_TrigMask_18
Trigger mask 18.

enumerator kBCTU_TrigMask_19
Trigger mask 19.

enumerator kBCTU_TrigMask_20
Trigger mask 20.

enumerator kBCTU_TrigMask_21
Trigger mask 21.

enumerator kBCTU_TrigMask_22
Trigger mask 22.

enumerator kBCTU_TrigMask_23
Trigger mask 23.

enumerator kBCTU_TrigMask_24
Trigger mask 24.

enumerator kBCTU_TrigMask_25
Trigger mask 25.

enumerator kBCTU_TrigMask_26
Trigger mask 26.

enumerator kBCTU_TrigMask_27
Trigger mask 27.

enumerator kBCTU_TrigMask_28
Trigger mask 28.

enumerator kBCTU_TrigMask_29
Trigger mask 29.

enumerator kBCTU_TrigMask_30
Trigger mask 30.

enumerator kBCTU_TrigMask_31
Trigger mask 31.

enum _bctu_trig_group
BCTU software trigger group.

Values:

enumerator kBCTU_TrigGroup_0
Trigger group 0.

enumerator kBCTU_TrigGroup_1
Trigger group 1.

enum _bctu_fifo
BCTU FIFO index.

Values:

enumerator kBCTU_Fifo_1
FIFO1

enumerator kBCTU_Fifo_2
FIFO2

enum _bctu_adc_data_dest
The destination for storing the conversion results.

Values:

enumerator kBCTU_DataDest_AdcReg
ADC-specific data registers.

enumerator kBCTU_DataDest_Fifo1
FIFO1.

enumerator kBCTU_DataDest_Fifo2
FIFO2.

enum _bctu_trig_conv_type
BCTU trigger conversion type.

Values:

enumerator kBCTU_TrigRes_SingleConv
Single conversion.

enumerator kBCTU_TrigRes_ConvList
Conversion list conversions.

enum _bctu_write_protect

Controls the protection of write-protected registers.

Values:

enumerator kBCTU_ProtectEn

Enable protection.

enumerator kBCTU_ProtectDis_OneWriteCycle

Disable protection for one write cycle.

enumerator kBCTU_ProtectDis_Permanent

Disable protection permanently.

typedef enum *_bctu_trig_group* bctu_trig_group_t

BCTU software trigger group.

typedef enum *_bctu_fifo* bctu_fifo_t

BCTU FIFO index.

typedef enum *_bctu_adc_data_dest* bctu_adc_data_dest_t

The destination for storing the conversion results.

typedef enum *_bctu_trig_conv_type* bctu_trig_conv_type_t

BCTU trigger conversion type.

typedef enum *_bctu_write_protect* bctu_write_protect_t

Controls the protection of write-protected registers.

typedef struct *_bctu_config* bctu_config_t

BCTU configuration.

typedef struct *_bctu_trig_config* bctu_trig_config_t

BCTU trigger configuration.

typedef struct *_bctu_convlist_config* bctu_convlist_config_t

BCTU conversion list configuration.

typedef struct *_bctu_adc_conv_result* bctu_adc_conv_result_t

ADC conversion result.

typedef struct *_bctu_fifo_res* bctu_fifo_res_t

BCTU fifo result.

struct _bctu_config

#include <fsl_bctu.h> BCTU configuration.

Public Members

bool debugFreezeEn

Disables all hardware trigger inputs but leaves the software trigger enabled. Debug Freeze isolates BCTU from external triggers and allows you to manually trigger a conversion and read the conversion result.

bctu_write_protect_t writeProtect

Specify the write protect mode.

struct _bctu_trig_config

#include <fsl_bctu.h> BCTU trigger configuration.

Public Members

bool enableLoop

Decides whether to enable current trigger executes in a loop.

uint8_t chanAddr

Sets ADC channel or a conversion list address.

uint8_t trigIndex

Specify trigger index. For hardware trigger, please use the member of enumeration @_bctu_trig_source. For software trigger, please specify a value directly and make sure that the value is within the allowed range. Generally speaking, the number of software triggers is the same as the number of hardware triggers.

uint32_t targetAdc

Sets which ADCs for conversion, _bctu_trig_adc.

bctu_trig_conv_type_t trigRes

Specify the trigger resolution.

bctu_adc_data_dest_t dataDest

Specify the destination for storing the conversion results.

struct __bctu_convlist_config

#include <fsl_bctu.h> BCTU conversion list configuration.

Public Members

bool lastChan

Specifies this element as the last channel in the conversion list.

bool lastChanPlusOne

Specifies this element as the next-to-last channel in the conversion list.

bool waitTrig

Instructs conversion list execution to stop after executing the current ADC channel. Execution begins again when the same trigger, which started the conversion list, reoccurs.

bool waitTrigPlusOne

Next Channel Wait For Trigger Plus 1.

uint8_t adcChan

Specifies the ADC channel to use.

uint8_t adcChanPlusOne

ADC Channel Selection Plus 1.

struct __bctu_adc_conv_result

#include <fsl_bctu.h> ADC conversion result.

Public Members

bool trigConvType

Indicates whether the conversion was part of a conversion list (0x1U)or a single conversion trigger (0x0U).

bool lastConv

For conversion list conversions, indicates this conversion result is the last conversion of the conversion list.

uint8_t trigSrc
: Contains the trigger number used to trigger the conversion.

uint8_t chanNum
: Contains the ADC channel used for the conversion.

uint16_t data
: Contains the data from the conversion.

struct __bctu_fifo_res
: *#include <fsl_bctu.h>* BCTU fifo result.

Public Members

uint8_t trigSrc
: Indicates the trigger number used to trigger the conversion..

uint8_t chanNum
: Indicates the ADC channel used for the conversion.

uint8_t adcNum
: Indicates the ADC used for conversion.

uint16_t convRes
: Contains the data from the conversion.

2.2 CACHE: ARMV7-M7 CACHE Memory Controller

static inline void L1CACHE_EnableICache(void)
: Enables cortex-m7 L1 instruction cache.

static inline void L1CACHE_DisableICache(void)
: Disables cortex-m7 L1 instruction cache.

static inline void L1CACHE_InvalidateICache(void)
: Invalidate cortex-m7 L1 instruction cache.

void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)
: Invalidate cortex-m7 L1 instruction cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 I-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The start address of the memory to be invalidated.
- size_byte – The memory size.

static inline void L1CACHE_EnableDCache(void)
: Enables cortex-m7 L1 data cache.

static inline void L1CACHE_DisableDCache(void)
: Disables cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCache(void)

Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_CleanDCache(void)

Cleans cortex-m7 L1 data cache.

static inline void L1CACHE_CleanInvalidateDCache(void)

Cleans and Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The start address of the memory to be invalidated.
- size_byte – The memory size.

static inline void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The start address of the memory to be cleaned.
- size_byte – The memory size.

static inline void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The start address of the memory to be clean and invalidated.
- size_byte – The memory size.

void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all instruction caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Cleans all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned.

void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned and invalidated.

FSL_CACHE_DRIVER_VERSION
cache driver version 2.0.4.

2.3 Clock Driver

enum _clock_ip_name
Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

enumerator kCLOCK_IpInvalid
Invalid Ip Name.

enumerator kCLOCK_Trgmux
Trigger Multiplexing Control (PRTN0_COFB1)

enumerator kCLOCK_Bctu
Body Cross Triggering Unit (PRTN0_COFB1)

enumerator kCLOCK_Emios0
EMIOS 0 (PRTN0_COFB1)

enumerator kCLOCK_Emios1
EMIOS 1 (PRTN0_COFB1)

enumerator kCLOCK_Lcu0
Logic Control Unit 0 (PRTN0_COFB1)

enumerator kCLOCK_Lcu1
Logic Control Unit 1 (PRTN0_COFB1)

enumerator kCLOCK_Adc0
Analog-to-digital converter 0 (PRTN0_COFB1)

enumerator kCLOCK_Adc1
Analog-to-digital converter 1 (PRTN0_COFB1)

enumerator kCLOCK_Pit0
Programmable Interrupt Timer 0 (PRTN0_COFB1)

enumerator kCLOCK_Pit1
Programmable Interrupt Timer 1 (PRTN0_COFB1)

enumerator kCLOCK_Edma
EDMA control & status (MP_CSR; MP_ES; MP_HRS) (PRTN1_COFB0)

enumerator kCLOCK_Tcd0
EDMA transfer control descriptor 0 (PRTN1_COFB0)

enumerator kCLOCK_Tcd1
EDMA transfer control descriptor 1 (PRTN1_COFB0)

enumerator kCLOCK_Tcd2
EDMA transfer control descriptor 2 (PRTN1_COFB0)

enumerator kCLOCK_Tcd3
EDMA transfer control descriptor 3 (PRTN1_COFB0)

enumerator kCLOCK_Tcd4
EDMA transfer control descriptor 4 (PRTN1_COFB0)

enumerator kCLOCK_Tcd5
EDMA transfer control descriptor 5 (PRTN1_COFB0)

enumerator kCLOCK_Tcd6
EDMA transfer control descriptor 6 (PRTN1_COFB0)

enumerator kCLOCK_Tcd7
EDMA transfer control descriptor 7 (PRTN1_COFB0)

enumerator kCLOCK_Tcd8
EDMA transfer control descriptor 8 (PRTN1_COFB0)

enumerator kCLOCK_Tcd9
EDMA transfer control descriptor 9 (PRTN1_COFB0)

enumerator kCLOCK_Tcd10
EDMA transfer control descriptor 10 (PRTN1_COFB0)

enumerator kCLOCK_Tcd11
EDMA transfer control descriptor 11 (PRTN1_COFB0)

enumerator kCLOCK_Sda
SDA-AP (PRTN1_COFB0)

enumerator kCLOCK_Eim
EIM (PRTN1_COFB0)

enumerator kCLOCK_Erm
ERM (PRTN1_COFB0)

enumerator kCLOCK_Mscm
MSCM (PRTN1_COFB0)

enumerator kCLOCK_Swt0
Software Watchdog 0 (PRTN1_COFB0)

enumerator kCLOCK_Stm0
System Timer Module 0 (PRTN1_COFB0)

enumerator kCLOCK_Intm
Interrupt Monitor (PRTN1_COFB0)

enumerator kCLOCK_Dmamux0
DMA Channel Multiplexer 0 (PRTN1_COFB1)

enumerator kCLOCK_Dmamux1
DMA Channel Multiplexer 1 (PRTN1_COFB1)

enumerator kCLOCK_Rtc
Real-time clock (PRTN1_COFB1)

enumerator kCLOCK_Vwrap
SIUL2_VIRTWRAPPER (PRTN1_COFB1)

enumerator kCLOCK_Wkup
Wakeup Unit (PRTN1_COFB1)

enumerator kCLOCK_Cmu05
CMU 0-5 (PRTN1_COFB1)

enumerator kCLOCK_Tspc
Touch Sensing Coupling Controller (PRTN1_COFB1)

enumerator kCLOCK_Sxosc
32 kHz Slow External Crystal Oscillator (PRTN1_COFB1)

enumerator kCLOCK_Fxosc
8-40 MHz Fast External Crystal Oscillator (PRTN1_COFB1)

enumerator kCLOCK_Pll
Frequency Modulated Phase-Locked Loop (PRTN1_COFB1)

enumerator kCLOCK_Flexcan0
FlexCAN 0 (PRTN1_COFB2)

enumerator kCLOCK_Flexcan1
FlexCAN 1 (PRTN1_COFB2)

enumerator kCLOCK_Flexcan2
FlexCAN 2 (PRTN1_COFB2)

enumerator kCLOCK_Flexcan3
FlexCAN 3 (PRTN1_COFB2)

enumerator kCLOCK_Flexcan4
FlexCAN 4 (PRTN1_COFB2)

enumerator kCLOCK_Flexcan5
FlexCAN 5 (PRTN1_COFB2)

enumerator kCLOCK_Flexio
Flexible IO (PRTN1_COFB2)

enumerator kCLOCK_Lpuart0
Low Power UART 0 (PRTN1_COFB2)

enumerator kCLOCK_Lpuart1
Low Power UART 1 (PRTN1_COFB2)

enumerator kCLOCK_Lpuart2
Low Power UART 2 (PRTN1_COFB2)

enumerator kCLOCK_Lpuart3
Low Power UART 3 (PRTN1_COFB2)

enumerator kCLOCK_Lpi2c0
Low Power I2C 0 (PRTN1_COFB2)

enumerator kCLOCK_Lpi2c1
Low Power I2C 1 (PRTN1_COFB2)

enumerator kCLOCK_Lpspi0
Low Power SPI 0 (PRTN1_COFB2)

enumerator kCLOCK_Lpspi1
Low Power SPI 1 (PRTN1_COFB2)

enumerator kCLOCK_Lpspi2
Low Power SPI 2 (PRTN1_COFB2)

enumerator kCLOCK_Lpspi3
Low Power SPI 3 (PRTN1_COFB2)

enumerator kCLOCK_Sai0
Synchronous Audio Interface 0 (PRTN1_COFB2)

enumerator kCLOCK_Lpcmp0
Low Power CMP 0 (PRTN1_COFB2)

enumerator kCLOCK_TempSensor
TMU Temperature Sensor Unit (PRTN1_COFB2)

enumerator kCLOCK_Crc0
CRC (PRTN1_COFB3)

enumerator kCLOCK_Tcd12
EDMA transfer control descriptor 12 (PRTN2_COFB0)

enumerator kCLOCK_Tcd13
EDMA transfer control descriptor 13 (PRTN2_COFB0)

enumerator kCLOCK_Tcd14
EDMA transfer control descriptor 14 (PRTN2_COFB0)

enumerator kCLOCK_Tcd15
EDMA transfer control descriptor 15 (PRTN2_COFB0)

enumerator kCLOCK_Tcd16
EDMA transfer control descriptor 16 (PRTN2_COFB0)

enumerator kCLOCK_Tcd17
EDMA transfer control descriptor 17 (PRTN2_COFB0)

enumerator kCLOCK_Tcd18
EDMA transfer control descriptor 18 (PRTN2_COFB0)

enumerator kCLOCK_Tcd19
EDMA transfer control descriptor 19 (PRTN2_COFB0)

enumerator kCLOCK_Tcd20
EDMA transfer control descriptor 20 (PRTN2_COFB0)

enumerator kCLOCK_Tcd21
EDMA transfer control descriptor 21 (PRTN2_COFB0)

enumerator kCLOCK_Tcd22
EDMA transfer control descriptor 22 (PRTN2_COFB0)

enumerator kCLOCK_Tcd23
EDMA transfer control descriptor 23 (PRTN2_COFB0)

enumerator kCLOCK_Tcd24
EDMA transfer control descriptor 24 (PRTN2_COFB0)

enumerator kCLOCK_Tcd25
EDMA transfer control descriptor 25 (PRTN2_COFB0)

enumerator kCLOCK_Tcd26
EDMA transfer control descriptor 26 (PRTN2_COFB0)

enumerator kCLOCK_Tcd27
EDMA transfer control descriptor 27 (PRTN2_COFB0)

enumerator kCLOCK_Tcd28
EDMA transfer control descriptor 28 (PRTN2_COFB0)

enumerator kCLOCK_Tcd29
EDMA transfer control descriptor 29 (PRTN2_COFB0)

enumerator kCLOCK_Tcd30
EDMA transfer control descriptor 30 (PRTN2_COFB0)

enumerator kCLOCK_Tcd31
EDMA transfer control descriptor 31 (PRTN2_COFB0)

enumerator kCLOCK_Sema42
Semaphores2 (PRTN2_COFB0)

enumerator kCLOCK_Stm1
System Timer Module 1 (PRTN2_COFB0)

enumerator kCLOCK_Emac
EMAC (PRTN2_COFB1)

enumerator kCLOCK_Lpuart8
Low Power UART 8 (PRTN2_COFB1).

enumerator kCLOCK_Lpuart9
Low Power UART 9 (PRTN2_COFB1).

enumerator kCLOCK_Lpuart10
Low Power UART 10 (PRTN2_COFB1).

enumerator kCLOCK_Lpuart11
Low Power UART 11 (PRTN2_COFB1).

enumerator kCLOCK_Lpuart12
Low Power UART 12 (PRTN2_COFB1).

enumerator kCLOCK_Lpuart13
Low Power UART 13 (PRTN2_COFB1).

enumerator kCLOCK_Lpuart14
Low Power UART 14 (PRTN2_COFB1).

enumerator kCLOCK_Lpuart15
Low Power UART 15 (PRTN2_COFB1).

enumerator kCLOCK_Lpspi4
Low Power SPI 4 (PRTN2_COFB1).

enumerator kCLOCK_Lpspi5
Low Power SPI 5 (PRTN2_COFB1).

enumerator kCLOCK_Qspi
QuadSPI (PRTN2_COFB1)

enumerator kCLOCK_Sai1
Synchronous Audio Interface 1 (PRTN2_COFB1)

enumerator kCLOCK_Lpcmp2
Low Power CMP 2 (PRTN2_COFB1)

enumerator kCLOCK_Tcm0
Core0 TCM (PRTN2_COFB1)

enumerator kCLOCK_Tcm1
Core1 TCM (PRTN2_COFB1)

enum _clock_div_name
Clock dividers.

Values:

enumerator kCLOCK_DivCoreClk
CLOCK MUX0, divider control 0.

enumerator kCLOCK_DivAipsPlatClk
CLOCK MUX0, divider control 1.

enumerator kCLOCK_DivAipsSlowClk
CLOCK MUX0, divider control 2.

enumerator kCLOCK_DivHseClk
CLOCK MUX0, divider control 3.

enumerator kCLOCK_DivDcmClk
CLOCK MUX0, divider control 4.

enumerator kCLOCK_DivLbistClk
CLOCK MUX0, divider control 5.

enumerator kCLOCK_DivStm0Clk
CLOCK MUX1, divider control 0.

enumerator kCLOCK_DivStm1Clk
CLOCK MUX2, divider control 0.

enumerator kCLOCK_DivFlexcan012PeClk
CLOCK MUX3, divider control 0.

enumerator kCLOCK_DivFlexcan345PeClk
CLOCK MUX4, divider control 0.

enumerator kCLOCK_DivClkoutStandbyClk
CLOCK MUX5, divider control 0.

enumerator kCLOCK_DivClkoutRunClk
CLOCK MUX6, divider control 0.

enumerator kCLOCK_DivTraceClk
CLOCK MUX11, divider control 0.

enum _clock_attach_id
The enumerator of clock attach Id.

Values:

enumerator kFIRC_CLK_to_MUX0
Select FIRC as CLOCK MUX0 clock source.

enumerator kPLL_PHI0_CLK_to_MUX0
Select PLL_PHI0_CLK as CLOCK MUX0 clock source.

enumerator kFIRC_CLK_to_STM0
Select FIRC as STM0 clock source.

enumerator kFXOSC_CLK_to_STM0
Select FXOSC as STM0 clock source.

enumerator kAIPS_PLAT_CLK_to_STM0
Select AIPS_PLAT_CLK as STM0 clock source.

enumerator kFIRC_CLK_to_STM1
Select FIRC as STM1 clock source.

enumerator kFXOSC_CLK_to_STM1
Select FXOSC as STM1 clock source.

enumerator kAIPS_PLAT_CLK_to_STM1
Select AIPS_PLAT_CLK as STM1 clock source.

enumerator kFIRC_CLK_to_FLEXCAN012_PE
Select FIRC as FLEXCAN0,1,2_PE clock source.

enumerator kFXOSC_CLK_to_FLEXCAN012_PE
Select FXOSC as FLEXCAN0,1,2_PE clock source.

enumerator kAIPS_PLAT_CLK_to_FLEXCAN012_PE
Select AIPS_PLAT_CLK as FLEXCAN0,1,2_PE clock source.

enumerator kFIRC_CLK_to_FLEXCAN345_PE
Select FIRC as FLEXCAN3,4,5_PE clock source.

enumerator kFXOSC_CLK_to_FLEXCAN345_PE
Select FXOSC as FLEXCAN3,4,5_PE clock source.

enumerator kAIPS_PLAT_CLK_to_FLEXCAN345_PE
Select AIPS_PLAT_CLK as FLEXCAN3,4,5_PE clock source.

enumerator kFIRC_CLK_to_CLKOUT_STANDBY
Select FIRC as CLKOUT_STANDBY mode clock source.

enumerator kSIRC_CLK_to_CLKOUT_STANDBY
Select SIRC as CLKOUT_STANDBY mode clock source.

enumerator kFXOSC_CLK_to_CLKOUT_STANDBY
Select FXOSC as CLKOUT_STANDBY mode clock source.

enumerator kSXOSC_CLK_to_CLKOUT_STANDBY
Select SXOSC as CLKOUT_STANDBY mode clock source.

enumerator kAIPS_SLOW_CLK_to_CLKOUT_STANDBY
Select AIPS_SLOW_CLK as CLKOUT_STANDBY mode clock source.

enumerator kFIRC_CLK_to_CLKOUT_RUN
Select FIRC as CLKOUT_RUN mode clock source.

enumerator kSIRC_CLK_to_CLKOUT_RUN
Select SIRC as CLKOUT_RUN mode clock source.

enumerator kFXOSC_CLK_to_CLKOUT_RUN
Select FXOSC as CLKOUT_RUN mode clock source.

enumerator kSXOSC_CLK_to_CLKOUT_RUN
Select SXOSC as CLKOUT_RUN mode clock source.

enumerator kPLL_PHI0_CLK_to_CLKOUT_RUN
Select PLL_PHI0_CLK as CLKOUT_RUN mode clock source.

enumerator kPLL_PHI1_CLK_to_CLKOUT_RUN
Select PLL_PHI1_CLK as CLKOUT_RUN mode clock source.

enumerator kCORE_CLK_to_CLKOUT_RUN
Select CORE_CLK as CLKOUT_RUN mode clock source.

enumerator kHSE_CLK_to_CLKOUT_RUN
Select HSE_CLK as CLKOUT_RUN mode clock source.

enumerator kAIPS_PLAT_CLK_to_CLKOUT_RUN
Select AIPS_PLAT as CLKOUT_RUN mode clock source.

enumerator kAIPS_SLOW_CLK_to_CLKOUT_RUN
Select AIPS_SLOW as CLKOUT_RUN mode clock source.

enumerator kFIRC_CLK_to_TRACE
Select FIRC as TRACE clock source.

enumerator kFXOSC_CLK_to_TRACE
Select FXOSC as TRACE clock source.

enumerator kPLL_PHI0_CLK_to_TRACE
Select PLL_PHI0_CLK as TRACE clock source.

enumerator kPLL_PHI1_CLK_to_TRACE
Select PLL_PHI1_CLK as TRACE clock source.

enum _clock_name
Clock name.

Values:

enumerator kCLOCK_CoreSysClk
Core clock.

enumerator kCLOCK_AipsPlatClk
AIPS_PLAT clock.

enumerator kCLOCK_AipsSlowClk
AIPS_SLOW clock.

enumerator kCLOCK_HseClk
HSE clock.

enumerator kCLOCK_DcmClk
DCM clock.

enumerator kCLOCK_LbistClk
LBIST clock.

enumerator kCLOCK_FircClk
Firc clock.

enumerator kCLOCK_SircClk
SIRC clock.

enumerator kCLOCK_FxoscClk
FXOSC clock.

enumerator kCLOCK_SxoscClk
SXOSC clock.

enumerator kCLOCK_PllPhi0Clk
PLL_PHI0_CLK clock.

enumerator kCLOCK_PllPhi1Clk
PLL_PHI0_CLK clock.

enumerator kCLOCK_Adc0Clk
ADC0 clock.

enumerator kCLOCK_Adc1Clk
ADC1 clock.

enumerator kCLOCK_BctuClk
Bctu clock.

enumerator kCLOCK_Cmp0Clk
CMP0 clock.

enumerator kCLOCK_EmiosClk
EMIOS clock.

enumerator kCLOCK_Flexcan0Clk
FLEXCAN0/1/2_PE clock.

enumerator kCLOCK_Flexcan1Clk
FLEXCAN0/1/2_PE clock.

enumerator kCLOCK_Flexcan2Clk
FLEXCAN0/1/2_PE clock.

enumerator kCLOCK_FlexioClk
FLEXIO clock.

enumerator kCLOCK_Lpi2c0Clk
LPI2C0 clock.

enumerator kCLOCK_Lpi2c1Clk
LPI2C0 clock.

enumerator kCLOCK_Lpspi0Clk
LPSPI0 clock.

enumerator kCLOCK_Lpspi1Clk
LPSPI1 clock.

enumerator kCLOCK_Lpspi2Clk
LPSPI2 clock.

enumerator kCLOCK_Lpspi3Clk
LPSPI3 clock.

enumerator kCLOCK_Lpuart0Clk
LPUART0 clock.

enumerator kCLOCK_Lpuart1Clk
LPUART1 clock.

enumerator kCLOCK_Lpuart2Clk
LPUART2 clock.

enumerator kCLOCK_Lpuart3Clk
LPUART3 clock.

enumerator kCLOCK_Pit0Clk
PIT0 clock.

enumerator kCLOCK_Pit1Clk
PIT1 clock.

enumerator kCLOCK_Sai0Clk
SAI0 clock.

enumerator kCLOCK_Sai1Clk
SAI0 clock.

enumerator kCLOCK_Stm0Clk
STM0 clock.

enumerator kCLOCK_QspiSfClk
QSPI_SF clock.

enumerator kCLOCK_EmacRxClk
EMAC RX clock.

enumerator kCLOCK_EmacTxClk
EMAC TX clock.

enumerator kCLOCK_EmacTsClk
EMAC TS clock.

enum __clock_trigger_div_type
Clock ip trigger divider type.

Values:

enumerator KCLOCK_ImmediateUpdate
Immediate divider update.

enumerator KCLOCK_CommonTriggerUpdate
Common trigger divider update.

enum __clock_switch_trigger_src
Clock switch trigger source.

Values:

enumerator kCLOCK_SwitchReserved
Reserved.

enumerator kCLOCK_SwitchSuccessPerReq
Common trigger divider update.

enumerator kCLOCK_SwitchFailedInactiveTarget
Switch after the request failed because of an inactive target clock and the current clock is FIRC.

enumerator kCLOCK_SwitchFailedInactiveCurrent
Switch after the request failed because of an inactive current clock and the current clock is FIRC.

enumerator kCLOCK_SwitchSafeReq
Switch to FIRC because of a safe clock request or reset succeeded.

enumerator kCLOCK_SwitchSafeReqInactivePreClk
Switch to FIRC because of a safe clock request or reset succeeded, but the previous current clock source was inactive.

enum clock_firc_div

The FIRC divider.

Values:

enumerator kFIRC_DividedBy2

FIRC clock divided by 2.

enumerator kFIRC_DividedBy16

FIRC clock divided by 16.

enumerator kFIRC_Undivided

FIRC clock undivided.

enum _fxosc_mode

The FXOSC work mode.

Values:

enumerator kFXOSC_ModeCrystal

Crystal mode.

enumerator kFXOSC_ModeBypass

Use external clock.

enum pll_mode

The PLL work mode.

Values:

enumerator kPLL_ModeInteger

PLL operates in integer-only mode.

enumerator kPLL_ModeFractional

PLL operates in fractional mode.

enumerator kPLL_ModeSSCG

PLL operates in Frequency Modulation mode.

enum pll_unlock_accuracy

The PLL unlock accuracy.

Values:

enumerator kPLL_UnlockAccuracy9

Unlock range = Expected value deviates by 9 (recommended when PLLFM[SSCGBYP] = 1).

enumerator kPLL_UnlockAccuracy17

Unlock range = Expected value deviates by 17 (recommended when PLLFM[SSCGBYP] = 1).

enumerator kPLL_UnlockAccuracy33

Unlock range = Expected value deviates by 33.

enumerator kPLL_UnlockAccuracy5

Unlock range = Expected value deviates by 5.

typedef enum *_clock_ip_name* clock_ip_name_t

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

typedef enum *_clock_div_name* clock_div_name_t

Clock dividers.

typedef enum *_clock_attach_id* clock_attach_id_t

The enumerator of clock attach Id.

typedef enum *_clock_name* clock_name_t

Clock name.

typedef enum *_clock_trigger_div_type* clock_trigger_div_type_t

Clock ip trigger divider type.

typedef enum *_clock_switch_trigger_src* clock_switch_trigger_src_t

Clock switch trigger source.

typedef enum *clock_firc_div* clock_firc_div_t

The FIRC divider.

typedef enum *_fxosc_mode* fxosc_mode_t

The FXOSC work mode.

typedef struct *_fxosc_config* fxosc_config_t

OSC Initialization Configuration Structure.

Defines the configuration data structure to initialize the FXOSC.

typedef enum *pll_mode* pll_mode_t

The PLL work mode.

typedef enum *pll_unlock_accuracy* pll_unlock_accuracy_t

The PLL unlock accuracy.

typedef struct *_pll_config* pll_config_t

PLL Initialization Configuration Structure.

Defines the configuration data structure to initialize the PLL. When porting to a new board, set the following members

typedef struct *_clock_pcfs_config_t* clock_pcfs_config_t

Clock Source PCFS configuration structure.

volatile uint32_t g_xtal0Freq

driver feature definition

External XTAL0 (FXOSC) clock frequency.

The XTAL0/EXTAL0 (FXOSC) clock frequency in Hz. When the clock is set up, use the function CLOCK_InitFxosc to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
Set up the FXOSC
CLOCK_InitFxosc(...);
```

static inline void CLOCK_McmeEnterKey(void)

Starts the hardware processes for the partition(s) clock change sequence.

Returns

Nothing

status_t CLOCK_EnableClock(*clock_ip_name_t* clk)

Enable the clock for specific IP.

Parameters

- clk – : Clock to be enabled.

Returns

kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

status_t CLOCK_DisableClock(*clock_ip_name_t* clk)

Disable the clock for specific IP.

Parameters

- clk – : Clock to be disabled.

Returns

kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

static inline void CLOCK_SetClkMux0DivTriggerType(*clock_trigger_div_type_t* type)

Setup peripheral clock dividers trigger type. Selects whether the dividers associated with clock mux 0 are updated immediately on writing to the corresponding divider configuration register (referred to as immediate divider update) or only on writing to the MC_CGM_MUX_0_DIV_TRIG register (referred to as common trigger update)

Parameters

- div_name – : Clock divider name
- type – : divider trigger type

Returns

Nothing

static inline *status_t* CLOCK_CommonTriggerClkMux0DivUpdate(void)

Trigger MUX0 clock dividers update(referred to as common trigger update). When MC_CGM_MUX_0_DIV_TRIG_CTRL_TCTL = 1, use this API to trigger update for MUX0 dividers.

Returns

kStatus_Timeout for timeout, kStatus_Success for sucees operation.

status_t CLOCK_SetFircDiv(*clock_firc_div_t* divider)

Setup FIRC clock divider.

Parameters

- divider – Value to be divided. Defined by clock_firc_div_t.

Return values

- kStatus_Success – FIRC divider set succeed.
- kStatus_Fail – FIRC divider set failed.

status_t CLOCK_SetClkDiv(*clock_div_name_t* div_name, uint32_t divider)

Setup peripheral clock dividers.

Parameters

- div_name – : Clock divider name
- divider – : Value to be divided. Divider value 0 will disable the divider. Undivided clock frequency / divider.

Returns

kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

static inline void CLOCK_EnableFircInStandbyMode(void)

Enable FIRC in standby mode.

Returns

Nothing

static inline void CLOCK_DisableFircInStandbyMode(void)

Disable FIRC in standby mode.

Returns
Nothing

static inline void CLOCK_EnableSircInStandbyMode(void)

Enable SIRC in standby mode.

Returns
Nothing

static inline void CLOCK_DisableSircInStandbyMode(void)

Disable SIRC in standby mode.

Returns
Nothing

status_t CLOCK_AttachClk(*clock_attach_id_t* connection)

Configure the clock selection muxes.

Parameters

- connection – : Clock to be configured.

Returns
kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

status_t CLOCK_SelectSafeClock(*clock_attach_id_t* connection)

Request safe clock switch to FIRC.

Parameters

- connection – : Clock to be configured.

Returns
kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

status_t CLOCK_ProgressiveClockFrequencySwitch(*clock_attach_id_t* connection, *clock_pcfs_config_t* const *config)

Configure the Progressive Clock Frequency Switch(PCFS) for MC_CGM MUX0.

Parameters

- connection – : Clock to be configured. Only MUX_0 clock supports PCFS, kPLL_PHI0_CLK_to_MUX0.
- config – : PCFS configuration.

Returns
kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

uint32_t CLOCK_GetClkSelectState(*clock_attach_id_t* connection)

Get the clock selection status.

Parameters

- connection – : Clock to be configured.

Returns
clock selection status

uint32_t CLOCK_GetClkSwitchTriggerCause(*clock_attach_id_t* connection)

Get the clock switch trigger source for hardware-controlled selector.

Parameters

- connection – : Clock to be configured.

Returns
clock selection status clock_switch_trigger_src_t.

status_t CLOCK_InitFxosc(const *fxosc_config_t* *config)

Initialize the FXOSC.

```
fxosc_config_t config =
{
  .freqHz = 16000000U,
  .workMode = kFXOSC_ModeCrystal,
  .startupDelay = 49U,
  .overdriveProtect = 12U,
};

CLOCK_InitFxosc(&config);
```

Parameters

- config – The FXOSC configuration structure, fxosc_config_t.

Returns

kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

status_t CLOCK_InitSxosc(bool enable, uint8_t startupDelay)

Initialize the SXOSC.

Parameters

- enable – Enable or disable the SXOSC.
- startupDelay – The oscillator counter runs on a divided crystal clock (divide by 4) and counts up to 128 times the EOCV value (EOCV * 128).

Returns

kStatus_Sucess for suceessfully operation, kStatus_Timeout for timeout.

static inline void CLOCK_DinitFxosc(void)

Disable FXOSC.

status_t CLOCK_InitPll(const *pll_config_t* *config)

Initialize the PLL.

```
pll_config_t config =
{
  .workMode = kPLL_ModeInteger,
  .preDiv = 2U,
  .postDiv = 2U,
  .multiplier = 120U,
  .accuracy = kPLL_UnlockAccuracy9,
  .outDiv[0] = 3U,
  .outDiv[1] = 3U,
};

CLOCK_InitPll(&pllConfig);
```

Parameters

- config – The PLL configuration structure, pll_config_t.

static inline void CLOCK_DeinitPll(void)

Deinit and disable the PLL.

static inline bool CLOCK_IsPllLossOfLock(void)

Check whether the PLL is loss of lock.

Returns
true: Loss of lock is detected, false: No loss of lock detected.

uint32_t CLOCK_GetFreq(*clock_name_t* name)

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_name_t.

Parameters

- name – Clock names defined in clock_name_t

Returns
Clock frequency value in hertz

uint32_t CLOCK_GetCoreClkFreq(void)

Get the Core clock frequency.

Returns
core clock frequency in HZ.

uint32_t CLOCK_GetFircClkFreq(void)

Get the FIRC clock frequency.

Returns
FIRC frequency in HZ.

uint32_t CLOCK_GetFxoscFreq(void)

Get the FXOSC clock frequency.

Returns
FXOSC frequency in HZ.

uint32_t CLOCK_GetAipsPlatClkFreq(void)

Get the AIPS_PLAT_CLK clock frequency.

Returns
frequency in HZ.

uint32_t CLOCK_GetAipsSlowClkFreq(void)

Get the AIPS_SLOW_CLK clock frequency.

Returns
frequency in HZ.

uint32_t CLOCK_GetHseClkFreq(void)

Get the HSE_CLK clock frequency.

Returns
frequency in HZ.

uint32_t CLOCK_GetDcmClkFreq(void)

Get the DCM_CLK clock frequency.

Returns
frequency in HZ.

uint32_t CLOCK_GetLbistClkFreq(void)

Get the LBIST_CLK clock frequency.

Returns
frequency in HZ.

uint32_t CLOCK_GetPllPhiClkFreq(uint32_t index)

Get the PLL_PHI clock frequency.

Parameters

- index – The PHI index.

Returns

PLL_PHIx frequency in HZ.

uint32_t CLOCK_GetFlexcanPeClkFreq(uint32_t index)

Get the FLEXCAN PE clock frequency.

Parameters

- index – The FLEXCAN index.

Returns

frequency in HZ.

uint32_t CLOCK_GetStmClkFreq(uint32_t index)

Get the STM clock frequency.

Parameters

- index – The STM index.

Returns

frequency in HZ.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.2.0.

CLOCK_RETRY_TIMES

Retry times for waiting flag.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

CLOCK_FIRC_CLK_FREQ

CLOCK_SIRC_CLK_FREQ

CLOCK_SXOSC_CLK_FREQ

CLOCK_FIRC_CLK

FIRC clock

CLOCK_SIRC_CLK

SIRC clock

CLOCK_FXOSC_CLK

FXOSC clock

CLOCK_SXOSC_CLK

SXOSC clock

CLOCK_PLL_PHI0_CLK

PLL PHI0 clock

CLOCK_PLL_PHI1_CLK

PLL PHI1 clock

CLOCK_CORE_CLK

M7 core clock

CLOCK_HSE_CLK
: HSE clock

CLOCK_AIPS_PLAT_CLK
: SRAM/AXBS clock

CLOCK_AIPS_SLOW_CLK
: peripheral clock

CLOCK_CLKOUT_RUN_CLK
: Clock output in RUN mode

TEMPSENSOR_CLOCKS
: Clock ip name array for TEMPSENSE.

BCTU_CLOCKS
: Clock ip name array for BCTU.

ADC_CLOCKS
: Clock ip name array for ADC.

DMAMUX_CLOCKS
: Clock ip name array for DMAMUX.

EDMA_CLOCKS
: Clock ip name array for EDMA.

EMAC_CLOCKS
: Clock ip name array for EMAC.

EDMA_TCD_CLOCKS
: Clock ip name array for EDMA TCD.

EIM_CLOCKS
: Clock ip name array for EIM.

EMIOS_CLOCKS
: Clock ip name array for EMIOS.

ERM_CLOCKS
: Clock ip name array for ERM.

FLEXCAN_CLOCKS
: Clock ip name array for FLEXCAN.

FLEXIO_CLOCKS
: Clock ip name array for FLEXIO.

QSPI_CLOCKS
: Clock ip name array for QSPI.

LCU_CLOCKS
: Clock ip name array for LCU.

LPCMP_CLOCKS
: Clock ip name array for LPCMP.

LPI2C_CLOCKS
: Clock ip name array for LPI2C.

LPUART_CLOCKS
: Clock ip name array for LPUART.

LPSPI_CLOCKS
: Clock ip name array for LPSPI.

PIT_CLOCKS
: Clock ip name array for PIT.

SEMA42_CLOCKS
: Clock ip name array for SEMA42.

STM_CLOCKS
: Clock ip name array for STM.

SWT_CLOCKS
: Clock ip name array for SWT.

TSPC_CLOCKS
: Clock ip name array for TSPC.

MC_ME_TUPLE_PRTN_MASK

MC_ME_TUPLE_PRTN_SHIFT

MC_ME_TUPLE_BIT_MASK

MC_ME_COFB_TUPLE(reg, bit)
: Help macro, bit16 to bit 28 is COFB register offset, bit0 to bit4 is bitfield.

MC_ME_COFB_STAT_CLKEN_OFFSET

MC_ME_COFB_OFFSET(tuple)

MC_ME_PRTN_PCONF_REG(tuple)

MC_ME_PRTN_PUPD_REG(tuple)

MC_ME_COFB_CLKEN_REG(tuple)

MC_ME_COFB_STAT_REG(tuple)

MC_ME_COFB_CLKEN_BIT(tuple)

CLOCK_DIV_TUPLE(mux, dc)

CLOCK_TUPLE_DIV_DC_REG(tuple)

CLOCK_TUPLE_DIV_UPD_STAT_REG(tuple)

CLOCK_TUPLE_MUX_CSC_REG(tuple)

CLOCK_TUPLE_MUX_CSS_REG(tuple)

uint32_t freqHz
: Frequency in Herz.

fxosc_mode_t workMode
: FXOSC work mode setting.

uint8_t startupDelay
: Specifies the end-of-count. Runs on crystal clock divided by 4 and counts to startupDelay a.

uint8_t overdriveProtect

pll_mode_t workMode
: PLL work mode setting.

uint8_t preDiv
: Input Clock Predivider.

uint8_t postDiv
: VCO clock post divider for driving the PHI output clock.

uint8_t multiplier
: Multiplication factor applied to the reference frequency.

uint16_t fracLoopDiv
: Numerator Of Fractional Loop Division Factor.Value should less than 18432. Used for Fractional Mode only.

uint16_t stepSize
: For SSCG mode. Frequency Modulation Step Size.

uint16_t stepNum
: For SSCG mode. Number Of Steps Of Modulation Period Or Frequency Modulation.

pll_unlock_accuracy_t accuracy
: PLL unlock accuracy.

uint8_t outDiv[PLL_PLLODIV_COUNT]
: PLL Output Divider.

uint32_t maxAllowableIDDchange
: Maximum variation of current per time (mA/microsec) - max allowable IDD change is determined by the user's power supply design.

uint32_t stepDuration
: Step duration of each PCFS step (time per step in us).

uint32_t clkSrcFreq
: Frequency of the clock source from which ramp-down and to which ramp-up are processed.

struct __fxosc__config
: *#include <fsl_clock.h>* OSC Initialization Configuration Structure.

  Defines the configuration data structure to initialize the FXOSC.

struct __pll__config
: *#include <fsl_clock.h>* PLL Initialization Configuration Structure.

  Defines the configuration data structure to initialize the PLL. When porting to a new board, set the following members

struct __clock__pcfs__config__t
: *#include <fsl_clock.h>* Clock Source PCFS configuration structure.

2.4 CMU_FC: CMU_FC Driver

2.5 Cmu_fc

void CMU_FC_GetDefaultConfig(*cmu_fc_config_t* *config)

Initializes CMU_FC configure structure.

This function initializes the CMU_FC configure structure to default value. The default value are:

```
config->refClockCount = CMU_FC_RCCR_REF_CNT_MASK;
config->interruptEnable = kCMU_FC_LowerThanLowThrAsyncInterruptEnable | kCMU_FC_
↪HigherThanHighThrAsyncInterruptEnable;
```

See also:

cmu_fc_config_t

Parameters

- config – Pointer to CMU_FC config structure.

status_t CMU_FC_Init(CMU_FC_Type *base, const *cmu_fc_config_t* *config)

Initializes the CMU_FC.

This function configures the peripheral for basic operation.

Parameters

- base – CMU_FC peripheral base address
- config – The configuration of CMU_FC

Return values

- kStatus_Success – Successfully initialize CMU_FC.
- kStatus_Fail – Initialize failed, because the module can not be initialized when GCR[FCE] = 1.

void CMU_FC_Deinit(CMU_FC_Type *base)

Shuts down the CMU_FC.

Parameters

- base – CMU_FC peripheral base address

static inline void CMU_FC_ClearStatusFlags(CMU_FC_Type *base, uint32_t mask)

Clear status in SR register.

Parameters

- base – CMU_FC peripheral base address
- mask – Mask of CMU_FC status flags to clear.

static inline uint32_t CMU_FC_GetStatusFlags(CMU_FC_Type *base)

Get status in SR register.

Parameters

- base – CMU_FC peripheral base address

Returns

FLL, FHH and RS bit field in SR register

uint32_t CMU_FC_CalcMinRefClkCnt(uint32_t ref_clk, uint32_t bus_clk, uint32_t monitored_clk)

Calculate minimum reference clock count cycle.

Note: Higher values of reference count results in longer metering window, leading to better accuracy in metered clock measurement.

Parameters

- ref_clk – Reference clock frequency
- bus_clk – CMU_FC bus clock
- monitored_clk – The expected frequency of monitored clock

Returns

Minimum reference count

void CMU_FC_CalcOptimumThreshold(*cmu_fc_config_t* *config, uint32_t monitored_clk, float monitored_clk_deviation, uint32_t ref_clk, float ref_clk_deviation)

Calculate optimum high frequency reference threshold and low frequency reference threshold.

Parameters

- config – Pointer to a CMU_FC configuration structure
- monitored_clk – The expected frequency of monitored clock
- monitored_clk_deviation – The deviation of monitored clock
- ref_clk – Reference clock frequency
- ref_clk_deviation – The deviation of reference clock

static inline void CMU_FC_SetRefClkCnt(CMU_FC_Type *base, uint32_t cnt)

Set reference clock count Note: Writes to RCCR are disabled after GCR[FCE] = 1.

Parameters

- base – CMU_FC peripheral base address
- cnt – The reference clock count to be set.

static inline void CMU_FC_SetHighThresholdClkCnt(CMU_FC_Type *base, uint32_t cnt)

Set high reference value for the monitored clock. Note: Writes to HTCR are disabled after GCR[FCE] = 1.

Parameters

- base – CMU_FC peripheral base address
- cnt – The reference clock count to be set.

static inline void CMU_FC_SetLowThresholdClkCnt(CMU_FC_Type *base, uint32_t cnt)

Set low reference value for the monitored clock. Note: Writes to LTCR are disabled after GCR[FCE] = 1.

Parameters

- base – CMU_FC peripheral base address
- cnt – The reference clock count to be set.

static inline void CMU_FC_StartFreqChecking(CMU_FC_Type *base)

Start the frequency checking.

This function write value into CMU_FC_GCR_FCE register to enable the CMU_FC

Parameters

- base – CMU_FC peripheral base address

static inline void CMU_FC_StopFreqChecking(CMU_FC_Type *base)

Stop frequency checking.

This function write value into CMU_FC_GCR_FCE register to disable the CMU_FC.

Note: To stop the ongoing operation, write 0 to FCE only when SR[RS] = 1

Parameters

- base – CMU_FC peripheral base address

FSL_CMU_FC_DRIVER_VERSION

Defines CMU_FC driver version.

enum _cmu_fc_status_flags

List of CMU_FC status.

Values:

enumerator kCMU_FC_Running

Frequency check running

enumerator kCMU_FC_HigherThanHighThr

Frequency higher than high frequency reference threshold

enumerator kCMU_FC_LowerThanLowThr

Frequency lower than low frequency reference threshold

enum _cmu_fc_interrupt_flags

Values:

enumerator kCMU_FC_LowerThanLowThrInterruptEnable

Frequency Lower than Low Frequency Reference Threshold Synchronous Interrupt Enable

enumerator kCMU_FC_HigherThanHighThrInterruptEnable

Frequency Higher than High Frequency Reference Threshold Synchronous Interrupt Enable

enumerator kCMU_FC_LowerThanLowThrAsyncInterruptEnable

Frequency Lower than Low Frequency Reference Threshold Asynchronous Interrupt Enable

enumerator kCMU_FC_HigherThanHighThrAsyncInterruptEnable

Frequency Higher than High Frequency Reference Threshold Asynchronous Interrupt Enable

typedef enum *_cmu_fc_status_flags* cmu_fc_status_flags_t

List of CMU_FC status.

typedef enum *_cmu_fc_interrupt_flags* cmu_fc_interrupt_flags_t

typedef void (*cmu_fc_callback_t)(uint32_t flags)

Define CMU_FC interrupt callback function pointer.

void CMU_FC_DriverIRQHandler(uint32_t idx)

static inline void CMU_FC_EnableInterrupts(CMU_FC_Type *base, uint32_t mask)

Enable frequency check Interrupt Note: Writes to IER are disabled after GCR[FCE] = 1.

Parameters

- base – CMU_FC peripheral base address

static inline void CMU_FC_DisableInterrupts(CMU_FC_Type *base, uint32_t mask)

Disable frequency check Interrupt Note: Writes to IER are disabled after GCR[FCE] = 1.

Parameters

- base – CMU_FC peripheral base address

void CMU_FC_RegisterCallBack(CMU_FC_Type *base, *cmu_fc_callback_t* cb_func)

Register callback.

Parameters

- base – CMU_FC peripheral base address
- cb_func – callback function

struct cmu_fc_config_t

#include <fsl_cmu_fc.h> Describes CMU_FC configuration structure.

Public Members

uint32_t refClockCount

defines the duration of the checking operation in number of reference_clock cycles

uint32_t interruptEnable

Enable interrupt for specific event

uint32_t highThresholdCnt

Clock count for high frequency reference threshold of the monitored clock

uint32_t lowThresholdCnt

Clock count for lower frequency reference Threshold of the monitored clock

2.6 CMU_FM: CMU_FM Driver

void CMU_FM_GetDefaultConfig(*cmu_fm_config_t* *config)

Initializes CMU_FM configure structure.

This function initializes the CMU_FM configure structure to default value. The default value are:

```
config->refClockCount = CMU_FM_RCCR_REF_CNT_MASK;
config->enableInterrupt = true;
```

See also:

cmu_fm_config_t

Parameters

- config – Pointer to CMU_FM config structure.

status_t CMU_FM_Init(CMU_FM_Type *base, const *cmu_fm_config_t* *config)

Initializes the CMU_FM.

This function configures the peripheral for basic operation.

Parameters

- base – CMU_FM peripheral base address

- config – The configuration of CMU_FM

Return values

- kStatus_Success – Successfully initialize CMU_FM.
- kStatus_Fail – Initialize failed, because the module can not be initialized when GCR[FME] = 1.

void CMU_FM_Deinit(CMU_FM_Type *base)

Shuts down the CMU_FM.

Parameters

- base – CMU_FM peripheral base address

static inline void CMU_FM_ClearStatusFlags(CMU_FM_Type *base, uint32_t mask)

Clear status in SR register.

Parameters

- base – CMU_FM peripheral base address
- mask – Mask of CMU_FM status flags to clear.

static inline uint32_t CMU_FM_GetStatusFlags(CMU_FM_Type *base)

Get status in SR register.

Parameters

- base – CMU_FM peripheral base address

Returns

FMC, FMTO and RS bit field in SR register

uint32_t CMU_FM_CalcMinRefClkCnt(uint32_t ref_clk, uint32_t bus_clk, uint32_t monitored_clk)

Calculate minimum reference clock count cycle.

Note: Higher values of reference count results in longer metering window, leading to better accuracy in metered clock measurement.

Parameters

- ref_clk – Reference clock frequency
- bus_clk – CMU_FM bus clock
- monitored_clk – The expected frequency of monitored clock

Returns

Minimum reference count

static inline void CMU_FM_SetRefClkCnt(CMU_FM_Type *base, uint32_t cnt)

Set reference clock count Note: Writes to RCCR are disabled after GCR[FME] = 1.

Parameters

- base – CMU_FM peripheral base address
- cnt – The reference clock count to be set.

static inline uint32_t CMU_FM_GetMeteredClkCnt(CMU_FM_Type *base)

Obtain the metered clock count cycles.

Parameters

- base – CMU_FM peripheral base address
- cnt – metered clock count cycles

Returns
metered clock count cycles.

static inline void CMU_FM_StartFreqMetering(CMU_FM_Type *base)

Start the frequency metering.

This function write value into CMU_FM_GCR_FME register to enable the CMU_FM

Parameters

- base – CMU_FM peripheral base address

static inline void CMU_FM_StopFreqMetering(CMU_FM_Type *base)

Stop frequency metering.

This function write value into CMU_FM_GCR_FME register to disable the CMU_FM.

Note: To stop the ongoing operation, write 0 to FME only when SR[RS] = 1

Parameters

- base – CMU_FM peripheral base address

FSL_CMU_FM_DRIVER_VERSION

Defines CMU_FM driver version.

enum _cmu_fm_status_flags

List of CMU_FM status.

Values:

enumerator kCMU_FM_Running

Frequency meter running

enumerator kCMU_FM_MeterTimeout

Frequency meter time out

enumerator kCMU_FM_MeterComplete

Frequency meter complete

typedef enum *_cmu_fm_status_flags* cmu_fm_status_flags_t

List of CMU_FM status.

typedef void (*cmu_fm_callback_t)(uint32_t flags)

Define CMU_FM interrupt callback function pointer.

void CMU_FM_DriverIRQHandler(uint32_t idx)

static inline uint32_t CMU_FM_CalcMeteredClkFreq(uint32_t meteredClkCnt, uint32_t refClKCnt, uint32_t refClkFreq)

Calculate the frequency of the metered clock signal.

Parameters

- meteredClkCnt – The clock count cycles of metered clock
- refClKCnt – The clock count cycles of reference clock
- refClkFreq – The frequency of reference clock

Returns
The frequency of metered clock

static inline void CMU_FM_EnableInterrupts(CMU_FM_Type *base)

Enable frequency meter complete Interrupt Note: Writes to IER are disabled after GCR[FME] = 1.

Parameters

- base – CMU_FM peripheral base address

static inline void CMU_FM_DisableInterrupts(CMU_FM_Type *base)

Disable frequency meter complete Interrupt Note: Writes to IER are disabled after GCR[FME] = 1.

Parameters

- base – CMU_FM peripheral base address

void CMU_FM_RegisterCallBack(CMU_FM_Type *base, *cmu_fm_callback_t* cb_func)

Register callback.

Parameters

- base – CMU_FM peripheral base address
- cb_func – callback function

struct cmu_fm_config_t

#include <fsl_cmu_fm.h> Describes CMU_FM configuration structure.

Public Members

uint32_t refClockCount

defines the duration of the metering operation in number of reference_clock cycles

bool enableInterrupt

Enable/Disable frequency meter complete interrupt

2.7 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.0.5.

Current version: 2.0.5

Change log:

- 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.

static inline void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

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

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

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

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

Parameters

- config – CRC protocol configuration structure.

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

Writes data to the CRC module.

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

Parameters

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

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

Returns

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

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

Returns

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

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

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

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()

2.8 DCM: Device Configuration Module

void DCM_GetDcfScanReport(uint32_t slot, *dcm_dcf_scan_report_t* *report)
: Get DCF scan report.

  Parameters

  - slot – Slot number, should be less than DCM_DCMSRR_COUNT.
  - report – The DCF scan report.

static inline void DCM_ClearDcfScanReport(uint32_t slot)
: Clear DCF scan report.

  Parameters

  - slot – Slot number, should be less than DCM_DCMSRR_COUNT.

static inline *dcm_life_cycle_t* DCM_GetRealLifeCycle(void)
: Get real life cycle.

  Returns
  The real life cycle.

static inline *dcm_life_cycle_t* DCM_GetCurrentLifeCycle(void)
: Get current life cycle.

  Returns
  The current life cycle.

void DCM_GetLifeCycleScanStatus(*dcm_life_cycle_scan_status_t* *status)
: Get life cycle scan status.

  Parameters

  - status – The life cycle scan status.

void DCM_ClearLifeCycleScanStatus(void)
: Clear life cycle scan status.

FSL_DCM_DRIVER_VERSION
: DCM driver version.

enum _dcm_life_cycle
: Life cycle.

  Values:

  enumerator kDCM_LifeCycleFailureAnalysis
  : Failure analysis (FA)

enumerator kDCM_LifeCyclePreFailureAnalysis
Pre-Failure analysis (Pre-FA)

enumerator kDCM_LifeCycleOemProduction
OEM production

enumerator kDCM_LifeCycleCustomerDelivery
Customer delivery

enumerator kDCM_LifeCycleMcuProduction
MCU production

enumerator kDCM_LifeCycleInField
In field

enum _dcm_life_cycle_scan_marking
Life cycle scan marking.

Values:

enumerator kDCM_LifeCycleScanNotScanned
Not scanned yet

enumerator kDCM_LifeCycleScanActive
Marked as active

enumerator kDCM_LifeCycleScanInactive
Marked as inactive

enumerator kDCM_LifeCycleScanRegionErased
Region is erased/virgin

enumerator kDCM_LifeCycleScanUnknown
Marked as inactive by an unknown pattern

enumerator kDCM_LifeCycleScanTimeout
Scanning timed out

typedef struct *_dcm_dcf_scan_report* dcm_dcf_scan_report_t
DCF scan report.

typedef enum *_dcm_life_cycle* dcm_life_cycle_t
Life cycle.

typedef struct *_dcm_life_cycle_slost_scan_status* dcm_life_cycle_slot_scan_status_t
Life cycle slot scan status structure.

typedef struct *_dcm_life_cycle_scan_status* dcm_life_cycle_scan_status_t
Life cycle scan status structure.

DCM_LC_SLOT_COUNT
DCM life cycle slot count.

struct _dcm_dcf_scan_report
#include <fsl_dcm.h> DCF scan report.

Public Members

uint32_t address
Address

uint32_t location
: Load location

uint32_t flashError
: Flash memory error, 1 if there is error

uint32_t chipSideError
: Chip side error, 1 if there is error

uint32_t scanTimeout
: 1 if scan timeout.

struct _dcm_life_cycle_slost_scan_status
: *#include <fsl_dcm.h>* Life cycle slot scan status structure.

Public Members

uint8_t hasError
: 1 means error, 0 means no error.

uint8_t marking
: See _dcm_life_cycle_scan_marking

uint8_t hasEccError
: 1 means error, 0 means no error.

uint8_t hasFlashError
: 1 means error, 0 means no error.

struct _dcm_life_cycle_scan_status
: *#include <fsl_dcm.h>* Life cycle scan status structure.

2.9 DCM_GPR: Device Configuration Module General-Purpose Registers

static inline void DCM_GPR_DisableDebugModeForModule(uint32_t mask)
: Disable debug mode for module when CM7_0 enters debug mode.

  Parameters

  - mask – The mask of modules to be disabled debug mode. Use the OR'ed value of _disable_debug_mode_for_module.

static inline void DCM_GPR_StandbyEntryIoConfig(void)
: Controls the IO state before entering standby mode.

static inline void DCM_GPR_StandbyExitIoConfig(void)
: Controls the IO state after exiting standby mode.

void DCM_GPR_StandbyExitConfig(const *standby_exit_config_t* *config)
: Configure the standby exit.

  Parameters

  - config – The configuration of standby exit. standby_exit_config_t.

static inline void DCM_GPR_EnableSupplyVoltageMonitor(*internal_supply_monitor_source_t* source)

Enable the supply voltage monitoring by ADC.

Note: For divider sources, remeber to enable them before monitoring.

Parameters

- source – The source of voltage used by ADC for supply monitoring. internal_supply_monitor_source_t.

static inline void DCM_GPR_DisableSupplyVoltageMonitor(void)

Disable the supply voltage monitoring by ADC.

static inline void DCM_GPR_EnableVssLvMonitor(void)

Enable the VSS_LV divider.

static inline void DCM_GPR_DisableVssLvMonitor(void)

Disable the VSS_LV divider.

static inline void DCM_GPR_EnableHvADivider(void)

Enable the VSS_HV_A divider.

static inline void DCM_GPR_DisableHvADivider(void)

Disable the VSS_HV_A divider.

static inline void DCM_GPR_EnableHvBDivider(void)

Enable the VSS_HV_B divider.

static inline void DCM_GPR_DisableHvBDivider(void)

Disable the VSS_HV_B divider.

static inline void DCM_GPR_EnableV15Divider(void)

Enable the VDD_1.5_DIV divider.

static inline void DCM_GPR_DisableV15Divider(void)

FSL_DCM_GPR_DRIVER_VERSION

DCM_GPR driver version 2.0.0.

enum _disable_debug_mode_for_module

Disable debug mode for module.

Values:

enumerator kDCM_GPR_disableEdmaDebug

EDMA remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFccuDebug

FCCU remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLcu0Debug

LCU0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLcu1Debug

LCU1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableEmios0Debug

eMIOS0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableEmios1Debug
eMIOS1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableEmios2Debug
eMIOS2 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableRtcDebug
RTC remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableSwt0Debug
SWT0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableSwt1Debug
SWT1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableStm0Debug
STM0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableStm1Debug
STM1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disablePit0Debug
PIT0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disablePit1Debug
PIT1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disablePit2Debug
PIT2 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpspi0Debug
LPSPI0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpspi1Debug
LPSPI1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpspi2Debug
LPSPI2 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpspi3Debug
LPSPI3 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpspi4Debug
LPSPI4 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpspi5Debug
LPSPI5 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpi2c0Debug
LPI2C0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableLpi2c1Debug
LPI2C1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFlexioDebug
FLEXIO remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFlexcan0Debug
FLEXCAN0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFlexcan1Debug
FLEXCAN1 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFlexcan2Debug
FLEXCAN2 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFlexcan3Debug
FLEXCAN3 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFlexcan4Debug
FLEXCAN4 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableFlexcan5Debug
FLEXCAN5 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableSai0Debug
SAI0 remains functional and is not impacted when CM7_0 enters debug mode.

enumerator kDCM_GPR_disableSai1Debug
SAI1 remains functional and is not impacted when CM7_0 enters debug mode.

enum __internal_supply_monitor_source
The source of voltage used by ADC for supply monitoring.

Values:

enumerator kDCM_GPR_VDD_HV_A_DIV
VDD_HV_A divider supply monitoring.

enumerator kDCM_GPR_VDD_HV_B_DIV
VDD_HV_B divider supply monitoring.

enumerator kDCM_GPR_VDD_V15_DIV
VDD 1.5V divider supply monitoring.

enumerator kDCM_GPR_VDD_V25_OSC
VDD 2.5V supply monitoring.

enumerator kDCM_GPR_VDD_V11_PD1H
VDD 1.1V PD1 Hot point supply monitoring.

enumerator kDCM_GPR_VDD_V11_PD1C
VDD 1.1V PD1 Cold point supply monitoring.

enumerator kDCM_GPR_VDD_V11_PLL
VDD 1.1V PLL supply monitoring.

enumerator kDCM_GPR_VDD_V11_PD0
VDD 1.1V PD0 supply monitoring.

typedef enum *_internal_supply_monitor_source* internal_supply_monitor_source_t
The source of voltage used by ADC for supply monitoring.

typedef struct *_standby_exit_config* standby_exit_config_t
Standby exit configuration.

struct __standby_exit_config
#include <fsl_dcm_gpr.h> Standby exit configuration.

Public Members

bool bypassSircTriming
Bypass SIRC trimming.

bool bypassPmcTrimingAndRgmDcfLoading
Bypass PMC trimming and RGM DCF loading.

bool bypassFircTriming
Bypass FIRC trimming.

bool enableFastStandbyExit
Enable fast standby exit.

uint32_t fastStandbyExitBootAddress
Cortex-M7_0 base address of vector table to be used after exiting Standby mode.

2.10 DMAMUX: Direct Memory Access Multiplexer Driver

void DMAMUX_Init(DMAMUX_Type *base)
Initializes the DMAMUX peripheral.

This function ungates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

void DMAMUX_Deinit(DMAMUX_Type *base)
Deinitializes the DMAMUX peripheral.

This function gates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

static inline void DMAMUX_EnableChannel(DMAMUX_Type *base, uint32_t channel)
Enables the DMAMUX channel.

This function enables the DMAMUX channel.

Parameters

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

static inline void DMAMUX_DisableChannel(DMAMUX_Type *base, uint32_t channel)
Disables the DMAMUX channel.

This function disables the DMAMUX channel.

Note: The user must disable the DMAMUX channel before configuring it.

Parameters

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

static inline void DMAMUX_SetSource(DMAMUX_Type *base, uint32_t channel, int32_t source)
Configures the DMAMUX channel source.

Parameters

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

- source – Channel source, which is used to trigger the DMA transfer.User need to use the dma_request_source_t type as the input parameter.

static inline void DMAMUX_EnablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX period trigger.

This function enables the DMAMUX period trigger feature.

Parameters

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

static inline void DMAMUX_DisablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX period trigger.

This function disables the DMAMUX period trigger.

Parameters

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

static inline void DMAMUX_EnableAlwaysOn(DMAMUX_Type *base, uint32_t channel, bool enable)

Enables the DMA channel to be always ON.

This function enables the DMAMUX channel always ON feature.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- enable – Switcher of the always ON feature. “true” means enabled, “false” means disabled.

FSL_DMAMUX_DRIVER_VERSION

DMAMUX driver version 2.1.1.

DMAMUX_CHANNEL_ENDIAN_CONVERTn(channel)

Macro used for dmamux channel endian convert.

2.11 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 specifing the sign extend bit position. If 0, sign extension is disabled.

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

Set channel swap size.

Parameters

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

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

Set channel access type.

Parameters

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

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

Set channel request source.

Parameters

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

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

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

Parameters

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

Returns

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

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

Set channel master ID replication.

Parameters

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

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

Set channel security level.

Parameters

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

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

Sets all TCD registers to default values.

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

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

Note: This function enables the auto stop request feature.

Parameters

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

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

Configures the eDMA transfer attribute.

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

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

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

Parameters

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

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

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

Configures the eDMA minor offset feature.

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

Parameters

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

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

Configures the eDMA channel preemption feature.

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

Parameters

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

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

Sets the channel link for the eDMA transfer.

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

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

Parameters

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

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

Sets the bandwidth for the eDMA transfer.

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

Parameters

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

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

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

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

Parameters

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

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

Enables an async request for the eDMA transfer.

Parameters

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

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

Enables an auto stop request for the eDMA transfer.

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

Parameters

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

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

Enables the interrupt source for the eDMA transfer.

Parameters

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

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

Disables the interrupt source for the eDMA transfer.

Parameters

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

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

Configures the eDMA channel TCD major offset feature.

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

Parameters

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

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

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

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

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

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

Note: This function enables the auto stop request feature.

Parameters

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

void EDMA_TcdReset(*edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

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

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

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

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

Configures the eDMA TCD transfer attribute.

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

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

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

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

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

Parameters

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

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

Configures the eDMA TCD minor offset feature.

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

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

Parameters

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

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

Sets the channel link for the eDMA TCD.

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

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

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

Parameters

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

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

Sets the bandwidth for the eDMA TCD.

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

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

Parameters

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

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

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

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

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

Parameters

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

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

Sets the auto stop request for the eDMA TCD.

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

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

Parameters

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

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

Enables the interrupt source for the eDMA TCD.

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

Parameters

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

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

Disables the interrupt source for the eDMA TCD.

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

Parameters

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

void EDMA_TcdSetMajorOffsetConfig(*edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

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

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

Parameters

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

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

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

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

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

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

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

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

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

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

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

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

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

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

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

Parameters

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

Returns

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

static inline uint32_t EDMA_GetErrorStatusFlags(*EDMA_Type* *base)

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

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

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

Gets the eDMA channel status flags.

Parameters

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

Returns

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

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

Clears the eDMA channel status flags.

Parameters

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

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

Creates the eDMA handle.

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

Parameters

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

Return values

- kStatus_Success –
- kStatus_InvalidArgument –

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

Installs the TCDs memory pool into the eDMA handle.

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

Parameters

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

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

Installs a callback function for the eDMA transfer.

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

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.

- userData – A parameter for the callback function.

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

Prepares the eDMA transfer structure configurations.

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

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

Parameters

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

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

Prepares the eDMA transfer structure.

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

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

Parameters

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

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

Prepares the eDMA transfer content descriptor.

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

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

Parameters

- handle – eDMA handle pointer.
- tcd – Pointer to eDMA transfer content descriptor structure.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- nextTcd – eDMA transfer linked TCD address.

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

Submits the eDMA transfer content descriptor.

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

Typical user case:

a. submit single transfer

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

b. submit static link transfer,

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

c. submit dynamic link transfer

```
edma_tcd_t tcdpool[2];
EDMA_InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
edma_tcd_t tcd;
```

(continues on next page)

(continued from previous page)

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

d. submit loop transfer

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

Parameters

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

Return values

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

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

Submits the eDMA transfer request.

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

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

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

status_t EDMA_SubmitLoopTransfer(*edma_handle_t* *handle, *edma_transfer_config_t* *transfer, uint32_t transferLoopCount)

Submits the eDMA scatter gather transfer configurations.

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
edma_channel_transfer_config_t transfer[2];
EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

---

Note: Application should check the return value of this function to avoid transfer request submit failed

---

Parameters

- handle – eDMA handle pointer
- transfer – pointer to user's eDMA channel configure structure, see edma_channel_transfer_config_t for detail
- transferLoopCount – the count of the transfer ring, if loop count is 1, that means that the one will link to itself.

Return values

- kStatus_Success – It means submit transfer request succeed
- kStatus_EDMA_Busy – channel is in busy status
- kStatus_InvalidArgument – Invalid Argument

void EDMA_StartTransfer(*edma_handle_t* *handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

void EDMA_StopTransfer(*edma_handle_t* *handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call EDMA_StartTransfer() again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

void EDMA_AbortTransfer(*edma_handle_t* *handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns

The next TCD address.

void EDMA_HandleIRQ(*edma_handle_t* *handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- handle – eDMA handle pointer.

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

Initialize all fields to 0 for the TCD structure.

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

Parameters

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

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.10.9.

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull

TCD queue is full.

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

enum _edma_transfer_size
eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes
Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes
Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes
Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes
Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes
Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes
Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes
Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA_TransferSize128Bytes
Source/Destination data transfer size is 128 bytes every time

enum _edma_modulo
eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable
Disable modulo

enumerator kEDMA_Modulo2bytes
Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes
Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes
Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes
Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes
Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone
No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle
eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type
Channel link type.

Values:

enumerator kEDMA_LinkNone
No channel link

enumerator kEDMA_MinorLink
Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.

Values:

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

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

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

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

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

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

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

_edma_interrupt_enable eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type
eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

enum edma_channel_memory_attribute
eDMA channel memory attribute

Values:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

enum _edma_channel_swap_size
eDMA4 channel swap size

Values:

enumerator kEDMA_ChannelSwapDisabled
Swap is disabled.

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

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

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

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

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

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

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

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

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

enum _edma_channel_access_type
eDMA4 channel access type

Values:

enumerator kEDMA_ChannelDataAccess
Data access for eDMA4 transfers.

enumerator kEDMA_ChannelInstructionAccess
Instruction access for eDMA4 transfers.

enum _edma_channel_protection_level
eDMA4 channel protection level

Values:

enumerator kEDMA_ChannelProtectionLevelUser
user protection level for eDMA transfers.

enumerator kEDMA_ChannelProtectionLevelPrivileged
Privileged protection level eDMA transfers.

typedef enum *_edma_transfer_size* edma_transfer_size_t
eDMA transfer configuration

typedef enum *_edma_modulo* edma_modulo_t
eDMA modulo configuration

typedef enum *_edma_bandwidth* edma_bandwidth_t
Bandwidth control.

typedef enum *_edma_channel_link_type* edma_channel_link_type_t
Channel link type.

typedef enum *_edma_transfer_type* edma_transfer_type_t
eDMA transfer type

typedef struct *_edma_channel_Preemption_config* edma_channel_Preemption_config_t
eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* edma_minor_offset_config_t
eDMA minor offset configuration

typedef enum *edma_channel_memory_attribute* edma_channel_memory_attribute_t
eDMA channel memory attribute

typedef enum *_edma_channel_swap_size* edma_channel_swap_size_t

eDMA4 channel swap size

typedef enum *_edma_channel_access_type* edma_channel_access_type_t

eDMA4 channel access type

typedef enum *_edma_channel_protection_level* edma_channel_protection_level_t

eDMA4 channel protection level

typedef struct *_edma_channel_config* edma_channel_config_t

eDMA4 channel configuration

typedef *edma_core_tcd_t* edma_tcd_t

eDMA TCD.

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

typedef struct *_edma_transfer_config* edma_transfer_config_t

edma4 channel transfer configuration

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

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

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

---

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

a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0

b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width

c. the totalBytes should align with the bytesEachRequest

d. the srcAddr should align with the srcWidthOfEachTransfer

e. the dstAddr should align with the dstWidthOfEachTransfer

f. the srcAddr should align with srcAddrModulo if modulo feature is enabled

g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.

---

typedef struct *_edma_config* edma_config_t

eDMA global configuration structure.

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

Define callback function for eDMA.

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

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

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

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

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

typedef struct *_edma_handle* edma_handle_t

eDMA transfer handle structure

FSL_EDMA_DRIVER_EDMA4

eDMA driver name

EDMA_ALLOCATE_TCD(name, number)

Macro used for allocate edma TCD.

DMA_DCHPRI_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct _edma_channel_Preemption_config

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption

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

bool enablePreemptAbility

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

uint8_t channelPriority

Channel priority

struct _edma_minor_offset_config

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

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

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

uint32_t minorOffset
: Offset for a minor loop mapping.

struct __edma__channel__config
: *#include <fsl_edma.h>* eDMA4 channel configuration

Public Members

edma_channel_Preemption_config_t channelPreemptionConfig
: channel preemption configuration

edma_channel_memory_attribute_t channelReadMemoryAttribute
: channel memory read attribute configuration

edma_channel_memory_attribute_t channelWriteMemoryAttribute
: channel memory write attribute configuration

edma_channel_swap_size_t channelSwapSize
: channel swap size configuration

edma_channel_access_type_t channelAccessType
: channel access type configuration

uint8_t channelDataSignExtensionBitPosition
: channel data sign extension bit psition configuration

uint32_t channelRequestSource
: hardware service request source for the channel

bool enableMasterIDReplication
: enable master ID replication

edma_channel_protection_level_t protectionLevel
: protection level

struct __edma__transfer__config
: *#include <fsl_edma.h>* edma4 channel transfer configuration

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

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

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

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

a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0

b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width

c. the totalBytes should align with the bytesEachRequest

d. the srcAddr should align with the srcWidthOfEachTransfer

e. the dstAddr should align with the dstWidthOfEachTransfer

f. the srcAddr should align with srcAddrModulo if modulo feature is enabled

g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

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

int16_t destOffset

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

uint32_t minorLoopBytes

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

uint32_t majorLoopCounts

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

uint16_t enabledInterruptMask

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

edma_modulo_t srcAddrModulo

source circular data queue range

int32_t srcMajorLoopOffset

source major loop offset

edma_modulo_t dstAddrModulo
: destination circular data queue range

int32_t dstMajorLoopOffset
: destination major loop offset

bool enableSrcMinorLoopOffset
: enable source minor loop offset

bool enableDstMinorLoopOffset
: enable dest minor loop offset

int32_t minorLoopOffset
: burst offset, the offset will be applied after minor loop update

bool enableChannelMajorLoopLink
: channel link when major loop complete

uint32_t majorLoopLinkChannel
: major loop link channel number

bool enableChannelMinorLoopLink
: channel link when minor loop complete

uint32_t minorLoopLinkChannel
: minor loop link channel number

edma_tcd_t *linkTCD
: pointer to the link transfer control descriptor

struct __edma__config
: *#include <fsl_edma.h>* eDMA global configuration structure.

Public Members

bool enableMasterIdReplication
: Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

bool enableGlobalChannelLink
: Enable(true) channel linking is available and controlled by each channel's link settings.

bool enableHaltOnError
: Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableDebugMode
: Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

bool enableRoundRobinArbitration
: Enable(true) channel linking is available and controlled by each channel's link settings.

edma_channel_config_t *channelConfig[1]
: channel preemption configuration

struct __edma__handle
: *#include <fsl_edma.h>* eDMA transfer handle structure

Public Members

edma_callback callback
Callback function for major count exhausted.

void *userData
Callback function parameter.

EDMA_ChannelType *channelBase
eDMA peripheral channel base address.

EDMA_Type *base
eDMA peripheral base address

EDMA_TCDType *tcdBase
eDMA peripheral tcd base address.

edma_tcd_t *tcdPool
Pointer to memory stored TCDs.

uint32_t channel
eDMA channel number.

volatile int8_t header
The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail
The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed
The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize
The total number of TCD slots in the queue.

2.12 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 interupt

DMA_DISABLE_ERROR_INT(base, channel)

DMA_CLEAR_ERROR_STATUS(base, channel)
: get/clear error status

DMA_GET_ERROR_STATUS(base, channel)

DMA_CLEAR_INT_STATUS(base, channel)
: get/clear INT status

DMA_GET_INT_STATUS(base, channel)

DMA_ENABLE_MAJOR_INT(base, channel)
: enable/dsiable MAJOR/HALF INT

DMA_ENABLE_HALF_INT(base, channel)

DMA_DISABLE_MAJOR_INT(base, channel)

DMA_DISABLE_HALF_INT(base, channel)

EDMA_TCD_ALIGN_SIZE
: EDMA tcd align size.

EDMA_CORE_BASE(base)
: EDMA base address convert macro.

EDMA_MP_BASE(base)

EDMA_CHANNEL_BASE(base, channel)

EDMA_TCD_BASE(base, channel)

EDMA_TCD_TYPE(x)

EDMA TCD type macro.

EDMA_TCD_SADDR(tcd, flag)

EDMA TCD address convert macro.

EDMA_TCD_SOFF(tcd, flag)

EDMA_TCD_ATTR(tcd, flag)

EDMA_TCD_NBYTES(tcd, flag)

EDMA_TCD_SLAST(tcd, flag)

EDMA_TCD_DADDR(tcd, flag)

EDMA_TCD_DOFF(tcd, flag)

EDMA_TCD_CITER(tcd, flag)

EDMA_TCD_DLAST_SGA(tcd, flag)

EDMA_TCD_CSR(tcd, flag)

EDMA_TCD_BITER(tcd, flag)

struct _edma_core_mp

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

Public Members

___IO uint32_t MP_CSR

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

___IO uint32_t MP_ES

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

struct _edma_core_channel

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

Public Members

___IO uint32_t CH_CSR

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

___IO uint32_t CH_ES

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

___IO uint32_t CH_INT

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

___IO uint32_t CH_SBR

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

___IO uint32_t CH_PRI

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

struct _edma5_core_tcd

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

Public Members

___IO uint32_t SADDR
SADDR register, used to save source address

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

___IO uint16_t SOFF
SOFF register, save offset bytes every transfer

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

___IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

___IO uint32_t SLAST
SLAST register

___IO uint32_t SLAST_SDA_HIGH
SLAST SDA HIGH register

___IO uint32_t DADDR
DADDR register, used for destination address

___IO uint32_t DADDR_HIGH
DADDR HIGH register, used for destination address

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

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

___IO uint16_t DOFF
DOFF register, used for destination offset

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

___IO uint16_t CSR
CSR register, for TCD control status

___IO uint16_t BITER
BITER register, begin minor loop count.

uint8_t RESERVED[16]
Aligned 64 bytes

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

Public Members

___IO uint32_t SADDR
SADDR register, used to save source address

___IO uint16_t SOFF
SOFF register, save offset bytes every transfer

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

___IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

___IO uint32_t SLAST
SLAST register

___IO uint32_t DADDR
DADDR register, used for destination address

___IO uint16_t DOFF
DOFF register, used for destination offset

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

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

___IO uint16_t CSR
CSR register, for TCD control status

___IO uint16_t BITER
BITER register, begin minor loop count.

struct _edma_core_tcd
#include <fsl_edma_core.h> edma core TCD struture definition

union MP_REGS

Public Members

struct *_edma_core_mp* EDMA5_REG

struct EDMA5_REG

Public Members

___IO uint32_t MP_INT_LOW
Channel Control and Status, array offset: 0x10008, array step: 0x10000

___I uint32_t MP_INT_HIGH
Channel Control and Status, array offset: 0x1000C, array step: 0x10000

___I uint32_t MP_HRS_LOW
Channel Control and Status, array offset: 0x10010, array step: 0x10000

___I 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

___I uint32_t MP_SSR_LOW
Channel Control and Status, array offset: 0x10030, array step: 0x10000

___I 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

2.13 eDMA soc Driver

FSL_EDMA_SOC_DRIVER_VERSION
Driver version 2.0.0.

FSL_EDMA_SOC_IP_DMA3
DMA IP version.

FSL_EDMA_SOC_IP_DMA4

EDMA_BASE_PTRS
DMA base table.

EDMA_CHN_IRQS

EDMA_CHANNEL_OFFSET

EDMA base address convert macro.

EDMA_CHANNEL_ARRAY_STEPn(base, channel)

2.14 eMIOS: Timer PWM Module

uint32_t EMIOS_GetInstance(EMIOS_Type *base)

Get the eMIOS instance from peripheral base address.

Parameters

- base – eMIOS peripheral base address.

Returns

eMIOS instance.

void EMIOS_Init(EMIOS_Type *base, const *emios_config_t* *config)

Initialize eMIOS instance.

Parameters

- base – eMIOS peripheral base address.
- config – Pointer to user's eMIOS config structure.

void EMIOS_Deinit(EMIOS_Type *base)

Disable eMIOS module and gate eMIOS clock.

Parameters

- base – eMIOS peripheral base address

void EMIOS_GetDefaultConfig(*emios_config_t* *config)

Gets the default configuration structure.

This function initializes the eMIOS configuration structure to default values. The default values are as follows. config->prescale = 0U; config->allowFreezeUC = false; config->useGlobalTimeBase = false;

Parameters

- config – Pointer to the eMIOS configuration structure.

void EMIOS_ConfigModulusCounter(EMIOS_Type *base, const *emios_uc_mc_config_t* *config, uint8_t channel)

Configure Unified Channel (UC) as Modulus Counter.

This function configures a channel to operate in Modulus Counter (MC) or Modulus Counter Buffered (MCB) mode. The counter can be configured for:

- Up count or Up/Down count modes
- Internal or external clock source
- Various reset timing options

Parameters

- base – eMIOS peripheral base address.
- config – Pointer to Modulus Counter configuration structure.
- channel – Channel number to configure.

void EMIOS_ConfigPWM(EMIOS_Type *base, const *emios_uc_pwm_config_t* *config, uint8_t channel)

Configure Unified Channel (UC) for PWM operation.

This function configures a channel for various PWM modes:

- OPWFMB: Output Pulse Width and Frequency Modulation Buffered
- OPWMCB: Center Aligned Output PWM with Dead Time Insertion Buffered
- OPWMB: Output PWM Buffered
- OPWMT: Output PWM with Trigger

Parameters

- base – eMIOS peripheral base address.
- config – Pointer to PWM configuration structure.
- channel – Channel number to configure.

void EMIOS_ConfigOutputCompare(EMIOS_Type *base, const *emios_uc_oc_config_t* *config, uint8_t channel)

Configure Unified Channel (UC) for Output Compare operation.

This function configures a channel for:

- SAOC: Single Action Output Compare
- DAOC: Double Action Output Compare

Parameters

- base – eMIOS peripheral base address.
- config – Pointer to Output Compare configuration structure.
- channel – Channel number to configure.

void EMIOS_ConfigInputCapture(EMIOS_Type *base, const *emios_uc_ic_config_t* *config, uint8_t channel)

Configure Unified Channel (UC) for Input Capture operation.

This function configures a channel for various input capture modes:

- SAIC: Single Action Input Capture
- IPWM: Input Pulse Width Measurement
- IPM: Input Period Measurement
- PEC: Pulse Edge Counting

Parameters

- base – eMIOS peripheral base address.
- config – Pointer to Input Capture configuration structure.
- channel – Channel number to configure.

void EMIOS_ConfigGPIO(EMIOS_Type *base, const *emios_uc_gpio_config_t* *config, uint8_t channel)

Configure Unified Channel (UC) for GPIO operation.

This function configures a channel for:

- GPIO Input or output mode
- Edge polarity for input mode

- Capture flag enable/disable

Parameters

- base – eMIOS peripheral base address.
- config – Pointer to GPIO configuration structure.
- channel – Channel number to configure.

void EMIOS_UpdatePWM(EMIOS_Type *base, const *emios_uc_pwm_config_t* *config, uint8_t channel)

Update PWM configuration for a channel.

This function updates the PWM parameters (duty cycle, period, etc.) for already configured PWM channel.

Parameters

- base – eMIOS peripheral base address.
- config – Pointer to PWM configuration structure.
- channel – Channel number to update.

static inline void EMIOS_SetAn(EMIOS_Type *base, uint32_t value, uint8_t channel)

Set Unified Channel (UC) A register value.

This function writes a value to the A register of a specified eMIOS channel. The A register is used for various purposes depending on the channel mode:

- In PWM modes: Stores duty cycle or leading edge position
- In input capture modes: Stores captured values
- In modulus counter modes: Stores period value

Parameters

- base – eMIOS peripheral base address.
- value – Value to write to the A register.
- channel – Channel number to configure.

static inline void EMIOS_SetBn(EMIOS_Type *base, uint32_t value, uint8_t channel)

Set Unified Channel (UC) B register value.

This function writes a value to the B register of a specified eMIOS channel. The B register is used for various purposes depending on the channel mode:

- In PWM modes: Stores period, dead time value or trailing edge position
- In input capture modes: Stores captured values

Parameters

- base – eMIOS peripheral base address.
- value – Value to write to the B register.
- channel – Channel number to configure.

static inline void EMIOS_SetALTAn(EMIOS_Type *base, uint32_t value, uint8_t channel)

Set Unified Channel (UC) Alternate A register value.

This function writes a value to the ALTA register of a specified eMIOS channel. The ALTA register is primarily used in PWM with Trigger (OPWMT) mode to store the trigger position value.

Parameters

- base – eMIOS peripheral base address.
- value – Value to write to the ALTA register.
- channel – Channel number to configure.

static inline void EMIOS_SetCNTn(EMIOS_Type *base, uint32_t value, uint8_t channel)

Set Unified Channel (UC) counter value.

This function writes a value to the internal counter of a specified eMIOS channel. The counter is used as a timebase in various modes. In modulus counter modes, writing to the counter can be used to initialize or reset the counting sequence.

Parameters

- base – eMIOS peripheral base address.
- value – Value to write to the counter.
- channel – Channel number to configure.

static inline uint32_t EMIOS_GetAn(EMIOS_Type *base, uint8_t channel)

Get Unified Channel (UC) A register value.

This function reads the current value from the A register of a specified eMIOS channel. The value returned depends on the channel's current mode of operation.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to read.

Returns

Current value of the A register (16-bit value).

static inline uint32_t EMIOS_GetBn(EMIOS_Type *base, uint8_t channel)

Get Unified Channel (UC) B register value.

This function reads the current value from the B register of a specified eMIOS channel. The value returned depends on the channel's current mode of operation.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to read.

Returns

Current value of the B register (16-bit value).

static inline uint32_t EMIOS_GetALTAn(EMIOS_Type *base, uint8_t channel)

Get Unified Channel (UC) Alternate A register value.

This function reads the current value from the ALTA register of a specified eMIOS channel. In Pulse Edge Counting mode, this returns number of detected pulses.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to read.

Returns

Current value of the ALTA register (16-bit value).

static inline void EMIOS_SetUCEdgePolarity(EMIOS_Type *base, uint8_t channel, uint32_t value)

Set Unified Channel (UC) edge polarity.

This function configures the edge polarity for a specified eMIOS channel. The edge polarity determines:

- For input modes: Which edge(s) trigger captures
- For output modes: Selects the logic level on the output pin

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.
- value – Edge polarity value depends on mode.

static inline void EMIOS_SetUCEdgeSelection(EMIOS_Type *base, uint8_t channel, uint32_t value)

Set Unified Channel (UC) edge selection.

This function configures the edge selection for a specified eMIOS channel. The edge selection determines:

- For input modes: Whether single or both edges trigger captures
- For SAOC mode: Selects the behavior of the output flip-flop at each match
- For GPIO in mode: Selects whether an edge asserts the input capture flag

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.
- value – Edge selection value depends on mode.

static inline void EMIOS_SetUCForceMatchA(EMIOS_Type *base, uint8_t channel, bool value)

Set Unified Channel (UC) force match A control.

This function forces a match on comparator A for the specified channel. When forced, the output behaves as if a match occurred on comparator A. This is useful for manually controlling output states in output modes.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.
- value – true to force match, false to clear force match.

static inline void EMIOS_SetUCForceMatchB(EMIOS_Type *base, uint8_t channel, bool value)

Set Unified Channel (UC) force match B control.

This function forces a match on comparator B for the specified channel. When forced, the output behaves as if a match occurred on comparator B. This is useful for manually controlling output states in output modes.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.
- value – true to force match, false to clear force match.

static inline void EMIOS_EnableUCPrescaler(EMIOS_Type *base, uint8_t channel)

Enable Unified Channel (UC) prescaler.

This function enables the channel's clock prescaler. The prescaler divides the global clock to provide the channel's timebase clock.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.

static inline void EMIOS_DisableUCPrescaler(EMIOS_Type *base, uint8_t channel)

Disable Unified Channel (UC) prescaler.

This function disables the channel's clock prescaler, stopping the channel's timebase clock.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.

static inline void EMIOS_EnableGlobalPrescaler(EMIOS_Type *base)

Enable eMIOS global prescaler.

This function enables the global clock prescaler that provides the base clock for all channels' prescalers.

Parameters

- base – eMIOS peripheral base address.

static inline void EMIOS_DisableGlobalPrescaler(EMIOS_Type *base)

Disable eMIOS global prescaler.

This function disables the global clock prescaler, stopping the base clock for all channels.

Parameters

- base – eMIOS peripheral base address.

static inline void EMIOS_SetChannelOutputUpdate(EMIOS_Type *base, uint32_t mask, bool enable)

Set channel output update control.

This function enables or disables output updates for the specified channels. When disabled, writes to channel registers don't affect the outputs until updates are re-enabled, allowing synchronized updates of multiple channels.

Parameters

- base – eMIOS peripheral base address.
- mask – Bitmask of channels to configure _emios_uc_index enumerations.
- enable – true to enable output updates, false to disable.

static inline void EMIOS_SetChannelStatus(EMIOS_Type *base, uint32_t mask, bool enable)

Enable or disable channels.

This function enables or disables the specified channels by stopping its clock.

Parameters

- base – eMIOS peripheral base address.
- mask – Bitmask of channels to configure _emios_uc_index enumerations.
- enable – true to enable channels, false to disable.

uint32_t EMIOS_GetCounterBusPeriod(EMIOS_Type *base, *emios_counterbus_t* source, uint8_t channel)

Get Unified Channel (UC) counter bus period.

Parameters

- base – eMIOS peripheral base address.
- source – Counter bus to query (A, BCD, or F).
- channel – Channel number associated with the counter bus.

Returns

Period value of the UC counter bus.

uint32_t EMIOS_GetCounterBusMode(EMIOS_Type *base, *emios_counterbus_t* source, uint8_t channel)

Get Unified Channel (UC) counter bus mode.

Parameters

- base – eMIOS peripheral base address.
- source – Counter bus to query (A, BCD, or F).
- channel – Channel number associated with the counter bus.

Returns

Mode of the UC counter bus (MC or MCB mode).

static inline void EMIOS_EnableUCInterrupt(EMIOS_Type *base, uint8_t channel)

Enable Unified Channel (UC) interrupt.

This function enables the interrupt generation for the specified channel. When enabled, the channel will generate interrupts on flag events (captures, compares, etc.) according to its configured mode.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.

static inline void EMIOS_EnableUCDMA(EMIOS_Type *base, uint8_t channel)

Enable Unified Channel (UC) DMA request.

This function enables DMA request generation for the specified channel. When enabled, the channel will generate DMA requests on flag events (captures, compares, etc.) according to its configured mode.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.

static inline void EMIOS_DisableUCInterruptAndDMA(EMIOS_Type *base, uint8_t channel)

Disable Unified Channel (UC) interrupt and DMA request.

This function disables both interrupt and DMA request generation for the specified channel.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to configure.

static inline uint32_t EMIOS_GetUCStatusFlag(EMIOS_Type *base, uint8_t channel)

Get Unified Channel (UC) status flags.

This function returns the current status flags for the specified channel. The flags indicate events such as captures, compares, overflows, etc.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to read.

Returns

Current status flags (combination of kEMIOS_EventFlag, kEMIOS_OverflowFlag, and kEMIOS_OverrunFlag).

static inline void EMIOS_ClearUCStatusFlag(EMIOS_Type *base, uint32_t mask, uint8_t channel)

Clear Unified Channel (UC) status flags.

This function clears the specified status flags for the given channel. Flags must be cleared to detect new events.

Parameters

- base – eMIOS peripheral base address.
- mask – Bitmask of flags to clear (combination of kEMIOS_EventFlag, kEMIOS_OverflowFlag, and kEMIOS_OverrunFlag).
- channel – Channel number to configure.

static inline uint32_t EMIOS_GetUCOutputPinStatus(EMIOS_Type *base, uint8_t channel)

Get Unified Channel (UC) output pin status.

This function returns the current state of the channel's output pin.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to read.

Returns

Current output pin state (0 = Negated, 1 = Asserted).

static inline uint32_t EMIOS_GetUCInputPinStatus(EMIOS_Type *base, uint8_t channel)

Get Unified Channel (UC) input pin status.

This function returns the current state of the channel's input pin.

Parameters

- base – eMIOS peripheral base address.
- channel – Channel number to read.

Returns

Current input pin state (0 = Negated, 1 = Asserted).

FSL_EMIOS_DRIVER_VERSION

eMIOS driver version 2.0.0.

enum _emios_uc_index

List of eMIOS UC index used in logic OR.

Values:

enumerator kEMIOS_UC0_Mask
UC 0 Mask

enumerator kEMIOS_UC1_Mask
UC 1 Mask

enumerator kEMIOS_UC2_Mask
UC 2 Mask

enumerator kEMIOS_UC3_Mask
UC 3 Mask

enumerator kEMIOS_UC4_Mask
UC 4 Mask

enumerator kEMIOS_UC5_Mask
UC 5 Mask

enumerator kEMIOS_UC6_Mask
UC 6 Mask

enumerator kEMIOS_UC7_Mask
UC 7 Mask

enumerator kEMIOS_UC8_Mask
UC 8 Mask

enumerator kEMIOS_UC9_Mask
UC 9 Mask

enumerator kEMIOS_UC10_Mask
UC 10 Mask

enumerator kEMIOS_UC11_Mask
UC 11 Mask

enumerator kEMIOS_UC12_Mask
UC 12 Mask

enumerator kEMIOS_UC13_Mask
UC 13 Mask

enumerator kEMIOS_UC14_Mask
UC 14 Mask

enumerator kEMIOS_UC15_Mask
UC 15 Mask

enumerator kEMIOS_UC16_Mask
UC 16 Mask

enumerator kEMIOS_UC17_Mask
UC 17 Mask

enumerator kEMIOS_UC18_Mask
UC 18 Mask

enumerator kEMIOS_UC19_Mask
UC 19 Mask

enumerator kEMIOS_UC20_Mask
UC 20 Mask

enumerator kEMIOS_UC21_Mask
UC 21 Mask

enumerator kEMIOS_UC22_Mask
UC 22 Mask

enumerator kEMIOS_UC23_Mask
UC 23 Mask

enum _emios_uc_status_flags
eMIOS Unified Channel (UC) status flags

Values:

enumerator kEMIOS_EventFlag
A match or capture event occurred

enumerator kEMIOS_OverflowFlag
Counter overflow occurred

enumerator kEMIOS_OverrunFlag
New event occurred before previous was serviced

enum _emios_uc_gpio_mode
eMIOS GPIO modes

Values:

enumerator kEMIOS_GPIO_Input
GPIO input mode.

enumerator kEMIOS_GPIO_Output
GPIO output mode.

enum _emios_uc_mc_mode
eMIOS Modulus Counter (MC) modes

Values:

enumerator kEMIOS_MC_UpCount
Modulus Counter Up mode

enumerator kEMIOS_MC_UpDownCount
Modulus Counter Up/Down mode

enumerator kEMIOS_MCB_UpCount
Modulus Counter Buffered Up mode

enumerator kEMIOS_MCB_UpDownCount_Flag
Modulus Counter Buffered Up/Down mode with flag on match start

enumerator kEMIOS_MCB_UpDownCount_FlagBoth
Modulus Counter Buffered Up/Down mode with flag on both edges

enum _emios_uc_pwm_mode
eMIOS PWM modes

Values:

enumerator kEMIOS_OPWFMB_Flag
OPWFMB mode with flag on BS1 match

enumerator kEMIOS_OPWFMB_FlagBoth
OPWFMB mode with flags on both matches

enumerator kEMIOS_OPWMCB_TrailEdge_Flag
OPWMCB mode with trailing edge dead time and flag

enumerator kEMIOS_OPWMCB_TrailEdge_FlagBoth
OPWMCB mode with trailing edge dead time and flag on both edges

enumerator kEMIOS_OPWMCB_LeadEdge_Flag
OPWMCB mode with leading edge dead time and flag

enumerator kEMIOS_OPWMCB_LeadEdge_FlagBoth
OPWMCB mode with leading edge dead time and flag on both edges

enumerator kEMIOS_OPWMB_Flag
OPWMB mode with flag on BS1 match

enumerator kEMIOS_OPWMB_FlagBoth
OPWMB mode with flag on both matches

enumerator kEMIOS_OPWMT
OPWMT mode with trigger generation

enum _emios_uc_oc_mode
eMIOS Output Compare (OC) modes

Values:

enumerator kEMIOS_SAOC
Single Action Output Compare mode

enumerator kEMIOS_DAOC_Flag
Double Action Output Compare mode with flag on B match

enumerator kEMIOS_DAOC_FlagBoth
Double Action Output Compare mode with flag on both matches

enum _emios_uc_ic_mode
eMIOS Input Capture (IC) modes

Values:

enumerator kEMIOS_SAIC
Single Action Input Capture mode

enumerator kEMIOS_SAIC_InputLevel
Single Action Input Capture mode with input level indication

enumerator kEMIOS_IPWM
Input Pulse Width Measurement mode

enumerator kEMIOS_IPM
Input Period Measurement mode

enumerator kEMIOS_PEC_Continuous
Pulse Edge Counting mode (continuous)

enumerator kEMIOS_PEC_SingleShot
Pulse Edge Counting mode (single shot)

enum _emios_clock_source
eMIOS clock source selection

Values:

enumerator kEMIOS_ClkSrcInternal
Use internal clock source (prescaled system clock)

enumerator kEMIOS_ClkSrcExternal
Use external clock source

enum _emios_counter_reset
eMIOS counter reset timing

Values:

enumerator kEMIOS_ResetMatchStart
Reset counter at match start

enumerator kEMIOS_ResetMatchEnd
Reset counter at match end

enum _emios_prescaler_source
eMIOS prescaler clock source

Values:

enumerator kEMIOS_Prescaler_PrescaledClock
Use prescaled clock (from global prescaler)

enumerator kEMIOS_Prescaler_ModuleClock
Use module clock (eMIOS module clock)

enum _emios_output_polarity
eMIOS output polarity

Values:

enumerator kEMIOS_Output_ActiveHigh
Active high output polarity

enumerator kEMIOS_Output_ActiveLow
Active low output polarity

enum _emios_counterbus
eMIOS counter bus selection

Values:

enumerator kEMIOS_CounterBus_A
Use global counter bus A (UC23)

enumerator kEMIOS_CounterBus_BCD
Use local counter bus B, C, or D (based on channel)

enumerator kEMIOS_CounterBus_F
Use global counter bus F (UC22)

enumerator kEMIOS_CounterBus_Internal
Use internal counter (channel's own counter)

enum _emios_output_disable_source
eMIOS output disable source selection

Values:

enumerator kEMIOS_OutputDisable_Source_0
Use output disable source 0

enumerator kEMIOS_OutputDisable_Source_1
Use output disable source 1

enumerator kEMIOS_OutputDisable_Source_2
Use output disable source 2

enumerator kEMIOS_OutputDisable_Source_3
Use output disable source 3

enumerator kEMIOS_OutputDisable_Source_None
No output disable source selected

enum _emios_oc_behavior
eMIOS output compare behavior

Values:

enumerator kEMIOS_OC_Set
Set output on match (to EDPOL value)

enumerator kEMIOS_OC_Toggle
Toggle output on match

enum _emios_ic_edge_select
eMIOS input capture edge selection

Values:

enumerator kEMIOS_IC_SingleEdge
Capture on single edge (rising or falling)

enumerator kEMIOS_IC_BothEdge
Capture on both edges

enum _emios_ic_edge_polarity
eMIOS input capture edge polarity

Values:

enumerator kEMIOS_IC_FallingEdge
Capture on falling edges

enumerator kEMIOS_IC_RisingEdge
Capture on rising edges

enum _emios_filter_source
eMIOS input filter clock source

Values:

enumerator kEMIOS_Filter_PrescaledClock
Use prescaled clock for input filter

enumerator kEMIOS_Filter_ModuleClock
Use module clock for input filter

enum _emios_filter_width
eMIOS input filter width

Values:

enumerator kEMIOS_Filter_Bypassed
Input filter bypassed

enumerator kEMIOS_Filter_2_Cycle
Minimum 2 clock cycles pulse width

enumerator kEMIOS_Filter_4_Cycle
Minimum 4 clock cycles pulse width

enumerator kEMIOS_Filter_8_Cycle
Minimum 8 clock cycles pulse width

enumerator kEMIOS_Filter_16_Cycle
Minimum 16 clock cycles pulse width

typedef enum *_emios_uc_index* emios_uc_index_t
List of eMIOS UC index used in logic OR.

typedef enum *_emios_uc_gpio_mode* emios_uc_gpio_mode_t
eMIOS GPIO modes

typedef enum *_emios_uc_mc_mode* emios_uc_mc_mode_t
eMIOS Modulus Counter (MC) modes

typedef enum *_emios_uc_pwm_mode* emios_uc_pwm_mode_t
eMIOS PWM modes

typedef enum *_emios_uc_oc_mode* emios_uc_oc_mode_t
eMIOS Output Compare (OC) modes

typedef enum *_emios_uc_ic_mode* emios_uc_ic_mode_t
eMIOS Input Capture (IC) modes

typedef enum *_emios_clock_source* emios_clock_source_t
eMIOS clock source selection

typedef enum *_emios_counter_reset* emios_counter_reset_t
eMIOS counter reset timing

typedef enum *_emios_prescaler_source* emios_prescaler_source_t
eMIOS prescaler clock source

typedef enum *_emios_output_polarity* emios_output_polarity_t
eMIOS output polarity

typedef enum *_emios_counterbus* emios_counterbus_t
eMIOS counter bus selection

typedef enum *_emios_output_disable_source* emios_output_disable_source_t
eMIOS output disable source selection

typedef enum *_emios_oc_behavior* emios_oc_behavior_t
eMIOS output compare behavior

typedef enum *_emios_ic_edge_select* emios_ic_edge_select_t
eMIOS input capture edge selection

typedef enum *_emios_ic_edge_polarity* emios_ic_edge_polarity_t
eMIOS input capture edge polarity

typedef enum *_emios_filter_source* emios_filter_source_t
eMIOS input filter clock source

typedef enum *_emios_filter_width* emios_filter_width_t
eMIOS input filter width

typedef struct *_emios_config* emios_config_t
: eMIOS module configuration structure.

typedef struct *_emios_uc_mc_config* emios_uc_mc_config_t
: Modulus Counter (MC) configuration structure. Configures channels in Modulus Counter or Modulus Counter Buffered modes.

typedef struct *_emios_uc_pwm_config* emios_uc_pwm_config_t
: PWM configuration structure. Configures channels in various PWM output modes.

typedef struct *_emios_uc_oc_config* emios_uc_oc_config_t
: Output Compare configuration structure. Configures channels in Single Action or Double Action Output Compare modes.

typedef struct *_emios_uc_ic_config* emios_uc_ic_config_t
: Input Capture configuration structure. Configures channels in various input capture modes.

typedef struct *_emios_uc_gpio_config* emios_uc_gpio_config_t
: GPIO configuration structure. Configures channels in General Purpose Input/Output mode.

UC_MC_MODE_MASK
: eMIOS Unified Channel (UC) Modulus Counter mode masks

  Mask for Modulus Counter modes

UC_MC_MODE_MC_UP
: Modulus Counter Up mode

UC_MC_MODE_MC_UPDOWN
: Modulus Counter Up/Down mode

UC_MC_MODE_MCB_UP
: Modulus Counter Buffered Up mode

UC_MC_MODE_MCB_UPDOWN
: Modulus Counter Buffered Up/Down mode

UC_PWM_MODE_MASK
: eMIOS Unified Channel (UC) PWM mode masks

  Mask for PWM modes

UC_PWM_MODE_OPWFMB
: Output Pulse Width and Frequency Modulation Buffered mode

UC_PWM_MODE_OPWMCB
: Center-aligned Output PWM with Dead Time Insertion Buffered mode

UC_PWM_MODE_OPWMB
: Output PWM Buffered mode

UC_PWM_MODE_OPWMT
: Output PWM with Trigger mode

UC_OC_MODE_MASK
: eMIOS Unified Channel (UC) Output Compare mode masks

  Mask for Output Compare modes

UC_OC_MODE_SAOC
: Single Action Output Compare mode

UC_OC_MODE_DAOC
: Double Action Output Compare mode

UC_IC_MODE_MASK
: eMIOS Unified Channel (UC) Input Capture mode masks

  Mask for Input Capture modes

UC_IC_MODE_SAIC
: Single Action Input Capture mode

UC_IC_MODE_IPWM
: Input Pulse Width Measurement mode

UC_IC_MODE_IPM
: Input Period Measurement mode

UC_IC_MODE_PECC
: Pulse Edge Counting Continuous mode

UC_IC_MODE_PECS
: Pulse Edge Counting Single-shot mode

EMIOS_COUNTERBUS_A
: eMIOS counter bus definitions

  Global counter bus A (driven by UC23)

EMIOS_COUNTERBUS_BCD
: Local counter buses B, C, D (driven by UC16, UC8, UC0)

EMIOS_COUNTERBUS_F
: Second global counter bus F (driven by UC22)

struct _emios_config
: *#include <fsl_emios.h>* eMIOS module configuration structure.

  Public Members

  uint8_t prescale
  : Global prescaler value from 1 (0) to 256 (0xFF).

  bool allowFreezeUC
  : Allow all UCs enter Freeze state when chip is in Debug mode.

  bool useGlobalTimeBase
  : Enable global timebase or disable.

struct _emios_uc_mc_config
: *#include <fsl_emios.h>* Modulus Counter (MC) configuration structure. Configures channels in Modulus Counter or Modulus Counter Buffered modes.

  Public Members

  uint32_t period
  : Counter period value written to register An directly.

  uint8_t prescale
  : Channel prescaler value 1 (0) to 16 (0xF)

uint8_t reloadOutputDelay
: Delay between reload signal assertions 1 (0) to 32 (0x1F)

emios_uc_mc_mode_t ucMode
: Counter mode

emios_clock_source_t clockSource
: Internal or external clock source.

emios_counter_reset_t counterResetTiming
: Reset counter at match start or end

emios_prescaler_source_t prescalerSource
: Clock source for prescaler.

emios_ic_edge_select_t edgeSelect
: Edge selection for external clock

emios_ic_edge_polarity_t edgePolarity
: Edge polarity for external clock

bool enableFreeze
: Allow channel to freeze in debug mode

struct _emios_uc_pwm_config
: *#include <fsl_emios.h>* PWM configuration structure. Configures channels in various PWM output modes.

Public Members

uint32_t period
: PWM period in ticks

uint32_t dutyCycle
: PWM duty cycle in ticks

uint32_t deadTime
: Dead time insertion value in ticks (for OPWMCB mode)

uint32_t phaseShift
: Phase shift value in ticks (for OPWMB OPWMT mode)

uint32_t triggerPosition
: Trigger position in ticks (for OPWMT mode)

uint8_t prescale
: Channel prescaler value 1 (0) to 16 (0xF) (for OPWFMB OPWMCB mode)

emios_uc_pwm_mode_t ucMode
: PWM mode

emios_counterbus_t counterBus
: Counter bus selection (A, BCD, F)

emios_output_polarity_t polarity
: Output active high or low

emios_prescaler_source_t prescalerSource
: Clock source for prescaler (for OPWFMB OPWMCB mode)

emios_output_disable_source_t outputDisableSource
Output disable source

bool enableFreeze
Allow channel to freeze in debug mode

struct __emios_uc_oc_config
#include <fsl_emios.h> Output Compare configuration structure. Configures channels in Single Action or Double Action Output Compare modes.

Public Members

uint32_t leadingEdge
Leading edge position in ticks

uint32_t trailingEdge
Trailing edge position in ticks (DAOC only)

emios_uc_oc_mode_t ucMode
Output Compare mode (SAOC, DAOC)

emios_counterbus_t counterBus
Counter bus selection (A, BCD, F)

emios_oc_behavior_t ocBehavior
Output behavior on match (Set or Toggle)

emios_output_polarity_t polarity
Output active high or low

bool enableFreeze
Allow channel to freeze in debug mode

struct __emios_uc_ic_config
#include <fsl_emios.h> Input Capture configuration structure. Configures channels in various input capture modes.

Public Members

uint32_t startTime
Start time for measurement window (PEC mode)

uint32_t endTime
End time for measurement window (PEC mode)

emios_uc_ic_mode_t ucMode
Input Capture mode

emios_counterbus_t counterBus
Counter bus selection (A, BCD, F)

emios_ic_edge_select_t edgeSelect
Single or both edges trigger

emios_ic_edge_polarity_t edgePolarity
Edge polarity for trigger

emios_filter_source_t filterClock
Clock source for input filter

emios_filter_width_t filterWidth
: Input filter width (bypass to 16 cycles)

bool enableFreeze
: Allow channel to freeze in debug mode

struct _emios_uc_gpio_config
: *#include <fsl_emios.h>* GPIO configuration structure. Configures channels in General Purpose Input/Output mode.

Public Members

emios_uc_gpio_mode_t ucMode
: GPIO mode (Input or Output)

emios_ic_edge_polarity_t edgePolarity
: Edge polarity for input mode

bool enableCaptureFlag
: Enable capture flag generation in input mode

2.15 EQOS-TSN: Ethernet QoS with TSN Driver

2.16 Enet_qos_qos

void ENET_QOS_GetDefaultConfig(*enet_qos_config_t* *config)
: Gets the ENET default configuration structure.

  The purpose of this API is to get the default ENET configure structure for ENET_QOS_Init(). User may use the initialized structure unchanged in ENET_QOS_Init(), or modify some fields of the structure before calling ENET_QOS_Init(). Example:

```
enet_qos_config_t config;
ENET_QOS_GetDefaultConfig(&config);
```

  Parameters

  - config – The ENET mac controller configuration structure pointer.

status_t ENET_QOS_Up(ENET_QOS_Type *base, const *enet_qos_config_t* *config, uint8_t *macAddr, uint8_t macCount, uint32_t refclkSrc_Hz)
: Initializes the ENET module.

  This function initializes it with the ENET basic configuration.

  Parameters

  - base – ENET peripheral base address.
  - config – ENET mac configuration structure pointer. The “enet_qos_config_t” type mac configuration return from ENET_QOS_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
  - macAddr – Pointer to ENET mac address array of Ethernet device. This MAC address should be provided.
  - macCount – Count of macAddr in the ENET mac address array

- refclkSrc_Hz – ENET input reference clock.

status_t ENET_QOS_Init(ENET_QOS_Type *base, const *enet_qos_config_t* *config, uint8_t *macAddr, uint8_t macCount, uint32_t refclkSrc_Hz)

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET basic configuration.

Parameters

- base – ENET peripheral base address.
- config – ENET mac configuration structure pointer. The "enet_qos_config_t" type mac configuration return from ENET_QOS_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- macAddr – Pointer to ENET mac address array of Ethernet device. This MAC address should be provided.
- macCount – Count of macAddr in the ENET mac address array
- refclkSrc_Hz – ENET input reference clock.

void ENET_QOS_Down(ENET_QOS_Type *base)

Stops the ENET module.

This function disables the ENET module.

Parameters

- base – ENET peripheral base address.

void ENET_QOS_Deinit(ENET_QOS_Type *base)

Deinitializes the ENET module.

This function gates the module clock and disables the ENET module.

Parameters

- base – ENET peripheral base address.

uint32_t ENET_QOS_GetInstance(ENET_QOS_Type *base)

Get the ENET instance from peripheral base address.

Parameters

- base – ENET peripheral base address.

Returns

ENET instance.

status_t ENET_QOS_DescriptorInit(ENET_QOS_Type *base, *enet_qos_config_t* *config, *enet_qos_buffer_config_t* *bufferConfig)

Initialize for all ENET descriptors.

Note: This function is do all tx/rx descriptors initialization. Because this API read all interrupt registers first and then set the interrupt flag for all descriptors, if the interrupt register is set. so the descriptor initialization should be called after ENET_QOS_Init(), ENET_QOS_EnableInterrupts() and ENET_QOS_CreateHandle()(if transactional APIs are used).

Parameters

- base – ENET peripheral base address.
- config – The configuration for ENET.

- bufferConfig – All buffers configuration.

status_t ENET_QOS_RxBufferAllocAll(ENET_QOS_Type *base, *enet_qos_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_QOS_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_QOS_CreateHandler() and buffer allocating callback function should be ready.

Parameters

- base – ENET_QOS peripheral base address.
- handle – The ENET_QOS handler structure. This is the same handler pointer used in the ENET_QOS_Init.

void ENET_QOS_RxBufferFreeAll(ENET_QOS_Type *base, *enet_qos_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_QOS peripheral base address.
- handle – The ENET_QOS handler structure. This is the same handler pointer used in the ENET_QOS_Init.

void ENET_QOS_StartRxTx(ENET_QOS_Type *base, uint8_t txRingNum, uint8_t rxRingNum)

Starts the ENET rx/tx. This function enable the tx/rx and starts the rx/tx 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_QOS_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_QOS_RING_NUM_MAX(2). If the ringNum is set with 1, the ring 0 will be used.

status_t ENET_QOS_SetMII(ENET_QOS_Type *base, *enet_qos_mii_speed_t* speed,
enet_qos_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.

Returns
kStatus_Success The ENET MII speed and duplex has been set successfully.

Returns
kStatus_InvalidArgument Could not set the desired ENET MII speed and duplex combination.

void ENET_QOS_SetSMI(ENET_QOS_Type *base, uint32_t csrClock_Hz)

Sets the ENET SMI(serial management interface)- MII management interface.

Parameters

- base – ENET peripheral base address.
- csrClock_Hz – CSR clock frequency in HZ

static inline bool ENET_QOS_IsSMIBusy(ENET_QOS_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_QOS_ReadSMIData(ENET_QOS_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_QOS_StartSMIWrite(ENET_QOS_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

Sends the MDIO IEEE802.3 Clause 22 format write command. After send command, user needs to check whether the transmission is over with ENET_QOS_IsSMIBusy().

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register address.
- data – The data written to PHY.

void ENET_QOS_StartSMIRead(ENET_QOS_Type *base, uint8_t phyAddr, uint8_t regAddr)

Sends the MDIO IEEE802.3 Clause 22 format read command. After send command, user needs to check whether the transmission is over with ENET_QOS_IsSMIBusy().

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address.
- regAddr – The PHY register address.

void ENET_QOS_StartExtC45SMIWrite(ENET_QOS_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)

Sends the MDIO IEEE802.3 Clause 45 format write command. After send command, user needs to check whether the transmission is over with ENET_QOS_IsSMIBusy().

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

void ENET_QOS_StartExtC45SMIRead(ENET_QOS_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr)

Sends the MDIO IEEE802.3 Clause 45 format read command. After send command, user needs to check whether the transmission is over with ENET_QOS_IsSMIBusy().

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.

status_t ENET_QOS_MDIOWrite(ENET_QOS_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

MDIO write with IEEE802.3 MDIO 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_QOS_MDIORead(ENET_QOS_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 MDIO 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.

status_t ENET_QOS_MDIOC45Write(ENET_QOS_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns
kStatus_Success MDIO access succeeds.

Returns
kStatus_Timeout MDIO access timeout.

status_t ENET_QOS_MDIOC45Read(ENET_QOS_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The data read from PHY.

Returns
kStatus_Success MDIO access succeeds.

Returns
kStatus_Timeout MDIO access timeout.

static inline void ENET_QOS_SetMacAddr(ENET_QOS_Type *base, uint8_t *macAddr, uint8_t index)

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.
- index – Configure macAddr to MAC_ADDRESS[index] register.

void ENET_QOS_GetMacAddr(ENET_QOS_Type *base, uint8_t *macAddr, uint8_t index)

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.
- index – Get macAddr from MAC_ADDRESS[index] register.

void ENET_QOS_AddMulticastGroup(ENET_QOS_Type *base, uint8_t *address)

Adds the ENET_QOS device to a multicast group.

Parameters

- base – ENET_QOS peripheral base address.
- address – The six-byte multicast group address which is provided by application.

void ENET_QOS_LeaveMulticastGroup(ENET_QOS_Type *base, uint8_t *address)

Moves the ENET_QOS device from a multicast group.

Parameters

- base – ENET_QOS peripheral base address.
- address – The six-byte multicast group address which is provided by application.

static inline void ENET_QOS_AcceptAllMulticast(ENET_QOS_Type *base)

Enable ENET device to accept all multicast frames.

Parameters

- base – ENET peripheral base address.

static inline void ENET_QOS_RejectAllMulticast(ENET_QOS_Type *base)

ENET device reject to accept all multicast frames.

Parameters

- base – ENET peripheral base address.

void ENET_QOS_EnterPowerDown(ENET_QOS_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_QOS_ExitPowerDown(ENET_QOS_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_QOS_EnableRxParser(ENET_QOS_Type *base, bool enable)

Enable/Disable Rx parser,please notice that for enable/disable Rx Parser, should better disable Receive first.

Parameters

- base – ENET_QOS peripheral base address.
- enable – Enable/Disable Rx parser function

Return values

- kStatus_Success – Configure rx parser success.
- kStatus_ENET_QOS_Timeout – Poll status flag timeout.

void ENET_QOS_EnableInterrupts(ENET_QOS_Type *base, uint32_t mask)

Enables the ENET DMA and MAC interrupts.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enet_qos_dma_interrupt_enable_t and enet_qos_mac_interrupt_enable_t. For example, to enable the dma and mac interrupt, do the following.

```
ENET_QOS_EnableInterrupts(ENET, kENET_QOS_DmaRx | kENET_QOS_DmaTx | kENET_
↪QOS_MacPmt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of both enumeration :: enet_qos_dma_interrupt_enable_t and enet_qos_mac_interrupt_enable_t.

void ENET_QOS_DisableInterrupts(ENET_QOS_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_qos_dma_interrupt_enable_t and enet_qos_mac_interrupt_enable_t. For example, to disable the dma and mac interrupt, do the following.

```
ENET_QOS_DisableInterrupts(ENET, kENET_QOS_DmaRx | kENET_QOS_DmaTx | kENET_
↪QOS_MacPmt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to disables. This is a logical OR of both enumeration :: enet_qos_dma_interrupt_enable_t and enet_qos_mac_interrupt_enable_t.

static inline uint32_t ENET_QOS_GetDmaInterruptStatus(ENET_QOS_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_QOS_RING_NUM_MAX.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: enet_qos_dma_interrupt_enable_t.

static inline void ENET_QOS_ClearDmaInterruptStatus(ENET_QOS_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_QOS_RING_NUM_MAX.
- mask – The interrupt status to be cleared. This is the logical OR of members of the enumeration :: enet_qos_dma_interrupt_enable_t.

static inline uint32_t ENET_QOS_GetMacInterruptStatus(ENET_QOS_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_qos_mac_interrupt_enable_t and right shift ENET_QOS_MACINT_ENUM_OFFSET to mask the returned value to get the exact interrupt status.

void ENET_QOS_ClearMacInterruptStatus(ENET_QOS_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_qos_mac_interrupt_enable_t. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_QOS_ClearMacInterruptStatus(ENET, kENET_QOS_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_qos_mac_interrupt_enable_t.

static inline bool ENET_QOS_IsTxDescriptorDmaOwn(*enet_qos_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_QOS_SetupTxDescriptor(*enet_qos_tx_bd_struct_t* *txDesc, void *buffer1, uint32_t bytes1, void *buffer2, uint32_t bytes2, uint32_t framelen, bool intEnable, bool tsEnable, *enet_qos_desc_flag* 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 requeued 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 fist buffer.
- buffer2 – The second buffer address in the descriptor.
- bytes2 – The bytes in the second buffer.
- framelen – The length of the frame to be transmitted.
- intEnable – Interrupt enable flag.
- tsEnable – The timestamp enable.
- flag – The flag of this tx descriptor, enet_qos_desc_flag .
- slotNum – The slot num used for AV only.

static inline void ENET_QOS_UpdateTxDescriptorTail(ENET_QOS_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_QOS_UpdateRxDescriptorTail(ENET_QOS_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_QOS_GetRxDescriptor(*enet_qos_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 initialization. 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_QOS_UpdateRxDescriptor(*enet_qos_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 initialization. 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.

status_t ENET_QOS_ConfigureRxParser(ENET_QOS_Type *base, *enet_qos_rxp_config_t* *rxpConfig, uint16_t entryCount)

Configure flexible rx parser.

This function is used to configure the flexible rx parser table.

Parameters

- base – ENET peripheral base address..
- rxpConfig – The rx parser configuration pointer.
- entryCount – The rx parser entry count.

Return values

- kStatus_Success – Configure rx parser success.
- kStatus_ENET_QOS_Timeout – Poll status flag timeout.

status_t ENET_QOS_ReadRxParser(ENET_QOS_Type *base, *enet_qos_rxp_config_t* *rxpConfig, uint16_t entryIndex)

Read flexible rx parser configuration at specified index.

This function is used to read flexible rx parser configuration at specified index.

Parameters

- base – ENET peripheral base address..
- rxpConfig – The rx parser configuration pointer.
- entryIndex – The rx parser entry index to read, start from 0.

Return values

- kStatus_Success – Configure rx parser success.
- kStatus_ENET_QOS_Timeout – Poll status flag timeout.

status_t ENET_QOS_EstProgramGcl(ENET_QOS_Type *base, *enet_qos_est_gcl_t* *gcl, uint32_t ptpClk_Hz)

Program Gate Control List.

This function is used to program the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

Parameters

- base – ENET peripheral base address..
- gcl – Pointer to the Gate Control List structure.
- ptpClk_Hz – frequency of the PTP clock.

status_t ENET_QOS_EstReadGcl(ENET_QOS_Type *base, *enet_qos_est_gcl_t* *gcl, uint32_t listLen, bool hwList)

Read Gate Control List.

This function is used to read the Enhanced Scheduled Transmisson list. (IEEE802.1Qbv)

Parameters

- base – ENET peripheral base address..
- gcl – Pointer to the Gate Control List structure.
- listLen – length of the provided opList array in gcl structure.
- hwList – Boolean if True read HW list, false read SW list.

static inline void ENET_QOS_FpeEnable(ENET_QOS_Type *base)

Enable Frame Preemption.

This function is used to enable frame preemption. (IEEE802.1Qbu)

Parameters

- base – ENET peripheral base address..

static inline void ENET_QOS_FpeDisable(ENET_QOS_Type *base)

Disable Frame Preemption.

This function is used to disable frame preemption. (IEEE802.1Qbu)

Parameters

- base – ENET peripheral base address..

static inline void ENET_QOS_FpeConfigPreemptable(ENET_QOS_Type *base, uint8_t queueMask)

Configure preemptable transmit queues.

This function is used to configure the preemptable queues. (IEEE802.1Qbu)

Parameters

- base – ENET peripheral base address..
- queueMask – bitmask representing queues to set in preemptable mode. The N-th bit represents the queue N.

void ENET_QOS_AVBConfigure(ENET_QOS_Type *base, const *enet_qos_cbs_config_t* *config, uint8_t queueIndex)

Sets the ENET AVB feature.

ENET_QOS AVB feature configuration, set transmit bandwidth. This API is called when the AVB feature is required.

Parameters

- base – ENET_QOS peripheral base address.
- config – The ENET_QOS AVB feature configuration structure.
- queueIndex – ENET_QOS queue index.

void ENET_QOS_GetStatistics(ENET_QOS_Type *base, *enet_qos_transfer_stats_t* *statistics)

Gets statistical data in transfer.

Parameters

- base – ENET_QOS peripheral base address.
- statistics – The statistics structure pointer.

void ENET_QOS_CreateHandler(ENET_QOS_Type *base, *enet_qos_handle_t* *handle, *enet_qos_config_t* *config, *enet_qos_buffer_config_t* *bufferConfig, *enet_qos_callback_t* callback, void *userData)

Create ENET Handler.

This is a transactional API and it's provided to store all data 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 function will store many data/flag for transactional use, so all configure API such as ENET_QOS_Init(), ENET_QOS_DescriptorInit(), ENET_QOS_EnableInterrupts() etc.

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_QOS_TxIntEvent with tx buffer free/requeue process APIs.

b. the tx interrupt is forced to open.

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_QOS_GetRxFrameSize(ENET_QOS_Type *base, *enet_qos_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_QOS_GetRxFrameSize, ENET_QOS_ReadFrame() should be called to update the receive buffers If the result is not "kStatus_ENET_QOS_RxFrameEmpty".

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_QOS_Init.
- length – The length of the valid frame received.
- channel – The DMAC channel for the rx.

Return values

- kStatus_ENET_QOS_RxFrameEmpty – No frame received. Should not call ENET_QOS_ReadFrame to read frame.
- kStatus_ENET_QOS_RxFrameError – Data error happens. ENET_QOS_ReadFrame should be called with NULL data and NULL length to update the receive buffers.
- kStatus_Success – Receive a frame Successfully then the ENET_QOS_ReadFrame should be called with the right data buffer and the captured data length input.

status_t ENET_QOS_ReadFrame(ENET_QOS_Type *base, *enet_qos_handle_t* *handle, uint8_t *data, uint32_t length, uint8_t channel, *enet_qos_ptp_time_t* *ts)

Reads a frame from the ENET device. This function reads a frame from the ENET DMA descriptors. The ENET_QOS_GetRxFrameSize should be used to get the size of the prepared data buffer. For example use rx dma channel 0:

```
uint32_t length;
enet_qos_handle_t g_handle;
status = ENET_QOS_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_QOS_ReadFrame(ENET, &g_handle, NULL, 0, 0);
    }
    else
    {
      status = ENET_QOS_ReadFrame(ENET, &g_handle, data, length, 0);
    }
}
else if (status == kStatus_ENET_QOS_RxFrameError)
{
    ENET_QOS_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_QOS_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.
- ts – Pointer to the structure enet_qos_ptp_time_t to save frame timestamp.

Returns

The execute status, successful or failure.

status_t ENET_QOS_SendFrame(ENET_QOS_Type *base, *enet_qos_handle_t* *handle, uint8_t *data, uint32_t length, uint8_t channel, bool isNeedTs, void *context, *enet_qos_tx_offload_t* txOffloadOps)

Transmits an ENET frame.

Note: The CRC is automatically appended to the data. Input the data to send without the CRC.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_QOS_Init.
- data – The data buffer provided by user to be send.
- length – The length of the data to be send.
- channel – Channel to send the frame, same with queue index.
- isNeedTs – True to enable timestamp save for the frame

- context – pointer to user context to be kept in the tx dirty frame information.
- txOffloadOps – The Tx frame checksum offload option.

Return values

- kStatus_Success – Send frame succeed.
- kStatus_ENET_QOS_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_QOS_TxFrameBusy.

void ENET_QOS_ReclaimTxDescriptor(ENET_QOS_Type *base, *enet_qos_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_QOS_Init.
- channel – The tx DMA channel.

void ENET_QOS_CommonIRQHandler(ENET_QOS_Type *base, *enet_qos_handle_t* *handle)

The ENET IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET_QOS_SetISRHandler(ENET_QOS_Type *base, *enet_qos_isr_t* ISRHandler)

Set the second level IRQ handler, allow user to overwrite the default second level weak IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

status_t ENET_QOS_Ptp1588CorrectTimerInCoarse(ENET_QOS_Type *base, *enet_qos_systime_op* operation, uint32_t second, uint32_t nanosecond)

Correct the ENET PTP 1588 timer in coarse method.

Parameters

- base – ENET peripheral base address.
- operation – The system time operation, refer to “enet_qos_systime_op”
- second – The correction second.
- nanosecond – The correction nanosecond.

status_t ENET_QOS_Ptp1588CorrectTimerInFine(ENET_QOS_Type *base, uint32_t addend)

Correct the ENET PTP 1588 timer in fine method.

Note: Should take refer to the chapter “System time correction” and see the description for the “fine correction method”.

Parameters

- base – ENET peripheral base address.
- addend – The addend value to be set in the fine method

static inline uint32_t ENET_QOS_Ptp1588GetAddend(ENET_QOS_Type *base)

Get the ENET Time stamp current addend value.

Parameters

- base – ENET peripheral base address.

Returns

The addend value.

void ENET_QOS_Ptp1588GetTimerNoIRQDisable(ENET_QOS_Type *base, uint64_t *second, uint32_t *nanosecond)

Gets the current ENET time from the PTP 1588 timer without IRQ disable.

Parameters

- base – ENET peripheral base address.
- second – The PTP 1588 system timer second.
- nanosecond – The PTP 1588 system timer nanosecond. For the unit of the nanosecond is 1ns. so the nanosecond is the real nanosecond.

static inline *status_t* ENET_Ptp1588PpsControl(ENET_QOS_Type *base, *enet_qos_ptp_pps_instance_t* instance, *enet_qos_ptp_pps_trgt_mode_t* trgtMode, *enet_qos_ptp_pps_cmd_t* cmd)

Sets the ENET PTP 1588 PPS control. All channels operate in flexible PPS output mode.

Parameters

- base – ENET peripheral base address.
- instance – The ENET QOS PTP PPS instance.
- trgtMode – The target time register mode.
- cmd – The target flexible PPS output control command.

status_t ENET_QOS_Ptp1588PpsSetTrgtTime(ENET_QOS_Type *base, *enet_qos_ptp_pps_instance_t* instance, uint32_t seconds, uint32_t nanoseconds)

Sets the ENET OQS PTP 1588 PPS target time registers.

Parameters

- base – ENET QOS peripheral base address.
- instance – The ENET QOS PTP PPS instance.
- seconds – The target seconds.
- nanoseconds – The target nanoseconds.

static inline void ENET_QOS_Ptp1588PpsSetWidth(ENET_QOS_Type *base, *enet_qos_ptp_pps_instance_t* instance, uint32_t width)

Sets the ENET OQS PTP 1588 PPS output signal interval.

Parameters

- base – ENET QOS peripheral base address.
- instance – The ENET QOS PTP PPS instance.

- width – Signal Width. It is stored in terms of number of units of sub-second increment value. The width value must be lesser than interval value.

static inline void ENET_QOS_Ptp1588PpsSetInterval(ENET_QOS_Type *base, *enet_qos_ptp_pps_instance_t* instance, uint32_t interval)

Sets the ENET OQS PTP 1588 PPS output signal width.

Parameters

- base – ENET QOS peripheral base address.
- instance – The ENET QOS PTP PPS instance.
- interval – Signal Interval. It is stored in terms of number of units of sub-second increment value.

void ENET_QOS_Ptp1588GetTimer(ENET_QOS_Type *base, uint64_t *second, uint32_t *nanosecond)

Gets the current ENET time from the PTP 1588 timer.

Parameters

- base – ENET peripheral base address.
- second – The PTP 1588 system timer second.
- nanosecond – The PTP 1588 system timer nanosecond. For the unit of the nanosecond is 1ns.so the nanosecond is the real nanosecond.

void ENET_QOS_GetTxFrame(*enet_qos_handle_t* *handle, *enet_qos_frame_info_t* *txFrame, uint8_t channel)

Gets the time stamp of the transmit frame.

This function is used for PTP stack to get the timestamp captured by the ENET driver.

Parameters

- handle – The ENET handler pointer.This is the same state pointer used in ENET_QOS_Init.
- txFrame – Input parameter, pointer to enet_qos_frame_info_t for saving read out frame information.
- channel – Channel for searching the tx frame.

status_t ENET_QOS_GetRxFrame(ENET_QOS_Type *base, *enet_qos_handle_t* *handle, *enet_qos_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. If user calls this function in Rx interrupt handler, be careful that this function makes Rx BD ready with allocating new buffer(normal) or updating current BD(out of memory). If there's always new Rx frame input, Rx interrupt will be triggered forever. Application need to disable Rx interrupt according to specific design in this case.

Parameters

- base – ENET peripheral base address.

- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- rxFrame – The received frame information structure provided by user.
- channel – Channel for searching the rx frame.

Return values

- kStatus_Success – Succeed to get one frame and allocate new memory for Rx buffer.
- kStatus_ENET_QOS_RxFrameEmpty – There's no Rx frame in the BD.
- kStatus_ENET_QOS_RxFrameError – There's issue in this receiving.
- kStatus_ENET_QOS_RxFrameDrop – There's no new buffer memory for BD, drop this frame.

FSL_ENET_QOS_DRIVER_VERSION

Defines the driver version.

ENET_QOS_RXDESCRIP_RD_BUFF1VALID_MASK

Defines for read format.

Buffer1 address valid.

ENET_QOS_RXDESCRIP_RD_BUFF2VALID_MASK

Buffer2 address valid.

ENET_QOS_RXDESCRIP_RD_IOC_MASK

Interrupt enable on complete.

ENET_QOS_RXDESCRIP_RD_OWN_MASK

Own bit.

ENET_QOS_RXDESCRIP_WR_ERR_MASK

Defines for write back format.

ENET_QOS_RXDESCRIP_WR_PYLOAD_MASK

ENET_QOS_RXDESCRIP_WR_PTPMSGTYPE_MASK

ENET_QOS_RXDESCRIP_WR_PTPTYPE_MASK

ENET_QOS_RXDESCRIP_WR_PTPVERSION_MASK

ENET_QOS_RXDESCRIP_WR_PTPTSA_MASK

ENET_QOS_RXDESCRIP_WR_PACKETLEN_MASK

ENET_QOS_RXDESCRIP_WR_ERRSUM_MASK

ENET_QOS_RXDESCRIP_WR_TYPE_MASK

ENET_QOS_RXDESCRIP_WR_DE_MASK

ENET_QOS_RXDESCRIP_WR_RE_MASK

ENET_QOS_RXDESCRIP_WR_OE_MASK

ENET_QOS_RXDESCRIP_WR_RWT_MASK

ENET_QOS_RXDESCRIP_WR_GP_MASK

ENET_QOS_RXDESCRIP_WR_CRC_MASK

ENET_QOS_RXDESCRIP_WR_RS0V_MASK

ENET_QOS_RXDESCRIP_WR_RS1V_MASK

ENET_QOS_RXDESCRIP_WR_RS2V_MASK

ENET_QOS_RXDESCRIP_WR_LD_MASK

ENET_QOS_RXDESCRIP_WR_FD_MASK

ENET_QOS_RXDESCRIP_WR_CTXT_MASK

ENET_QOS_RXDESCRIP_WR_OWN_MASK

ENET_QOS_RXDESCRIP_WR_SA_FAILURE_MASK

ENET_QOS_RXDESCRIP_WR_DA_FAILURE_MASK

ENET_QOS_TXDESCRIP_RD_BL1_MASK
: Defines for read format.

ENET_QOS_TXDESCRIP_RD_BL2_MASK

ENET_QOS_TXDESCRIP_RD_BL1(n)

ENET_QOS_TXDESCRIP_RD_BL2(n)

ENET_QOS_TXDESCRIP_RD_TTSE_MASK

ENET_QOS_TXDESCRIP_RD_IOC_MASK

ENET_QOS_TXDESCRIP_RD_FL_MASK

ENET_QOS_TXDESCRIP_RD_FL(n)

ENET_QOS_TXDESCRIP_RD_CIC(n)

ENET_QOS_TXDESCRIP_RD_TSE_MASK

ENET_QOS_TXDESCRIP_RD_SLOT(n)

ENET_QOS_TXDESCRIP_RD_SAIC(n)

ENET_QOS_TXDESCRIP_RD_CPC(n)

ENET_QOS_TXDESCRIP_RD_LDFD(n)

ENET_QOS_TXDESCRIP_RD_LD_MASK

ENET_QOS_TXDESCRIP_RD_FD_MASK

ENET_QOS_TXDESCRIP_RD_CTXT_MASK

ENET_QOS_TXDESCRIP_RD_OWN_MASK

ENET_QOS_TXDESCRIP_WB_TTSS_MASK
: Defines for write back format.

ENET_QOS_ABNORM_INT_MASK

ENET_QOS_NORM_INT_MASK

ENET_QOS_RING_NUM_MAX
: The Maximum number of tx/rx descriptor rings.

ENET_QOS_FRAME_MAX_FRAMELEN
Default maximum Ethernet frame size.

ENET_QOS_FCS_LEN
Ethernet FCS length.

ENET_QOS_ADDR_ALIGNMENT
Recommended Ethernet buffer alignment.

ENET_QOS_BUFF_ALIGNMENT
Receive buffer alignment shall be 4bytes-aligned.

ENET_QOS_MTL_RXFIFOSIZE
The rx fifo size.

ENET_QOS_MTL_TXFIFOSIZE
The tx fifo size.

ENET_QOS_MACINT_ENUM_OFFSET
The offest for mac interrupt in enum type.

ENET_QOS_RXP_ENTRY_COUNT
RXP table entry count, implied by FRPES in MAC_HW_FEATURE3

ENET_QOS_RXP_BUFFER_SIZE
RXP Buffer size, implied by FRPBS in MAC_HW_FEATURE3

ENET_QOS_EST_WID
Width of the time interval in Gate Control List

ENET_QOS_EST_DEP
Maxmimum depth of Gate Control List

Defines the status return codes for transaction.

Values:

enumerator kStatus_ENET_QOS_InitMemoryFail
Init fails since buffer memory is not enough.

enumerator kStatus_ENET_QOS_RxFrameError
A frame received but data error happen.

enumerator kStatus_ENET_QOS_RxFrameFail
Failed to receive a frame.

enumerator kStatus_ENET_QOS_RxFrameEmpty
No frame arrive.

enumerator kStatus_ENET_QOS_RxFrameDrop
Rx frame is dropped since no buffer memory.

enumerator kStatus_ENET_QOS_TxFrameBusy
Transmit descriptors are under process.

enumerator kStatus_ENET_QOS_TxFrameFail
Transmit frame fail.

enumerator kStatus_ENET_QOS_TxFrameOverLen
Transmit oversize.

enumerator kStatus_ENET_QOS_Est_SwListBusy
SW Gcl List not yet processed by HW.

enumerator kStatus_ENET_QOS_Est_SwListWriteAbort
SW Gcl List write aborted .

enumerator kStatus_ENET_QOS_Est_InvalidParameter
Invalid parameter in Gcl List .

enumerator kStatus_ENET_QOS_Est_BtrError
Base Time Error when loading list.

enumerator kStatus_ENET_QOS_TrgtBusy
Target time register busy.

enumerator kStatus_ENET_QOS_Timeout
Target time register busy.

enumerator kStatus_ENET_QOS_PpsBusy
Pps command busy.

enum _enet_qos_mii_mode
Defines the MII/RGMII mode for data interface between the MAC and the PHY.

Values:

enumerator kENET_QOS_MiiMode
MII mode for data interface.

enumerator kENET_QOS_RgmiiMode
RGMII mode for data interface.

enumerator kENET_QOS_RmiiMode
RMII mode for data interface.

enum _enet_qos_mii_speed
Defines the 10/100/1000 Mbps speed for the MII data interface.

Values:

enumerator kENET_QOS_MiiSpeed10M
Speed 10 Mbps.

enumerator kENET_QOS_MiiSpeed100M
Speed 100 Mbps.

enumerator kENET_QOS_MiiSpeed1000M
Speed 1000 Mbps.

enumerator kENET_QOS_MiiSpeed2500M
Speed 2500 Mbps.

enum _enet_qos_mii_duplex
Defines the half or full duplex for the MII data interface.

Values:

enumerator kENET_QOS_MiiHalfDuplex
Half duplex mode.

enumerator kENET_QOS_MiiFullDuplex
Full duplex mode.

enum _enet_qos_mii_normal_opcode

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

Values:

enumerator kENET_QOS_MiiWriteFrame

Write frame operation for a valid MII management frame.

enumerator kENET_QOS_MiiReadFrame

Read frame operation for a valid MII management frame.

enum _enet_qos_dma_burstlen

Define the DMA maximum transmit burst length.

Values:

enumerator kENET_QOS_BurstLen1

DMA burst length 1.

enumerator kENET_QOS_BurstLen2

DMA burst length 2.

enumerator kENET_QOS_BurstLen4

DMA burst length 4.

enumerator kENET_QOS_BurstLen8

DMA burst length 8.

enumerator kENET_QOS_BurstLen16

DMA burst length 16.

enumerator kENET_QOS_BurstLen32

DMA burst length 32.

enumerator kENET_QOS_BurstLen64

DMA burst length 64. eight times enabled.

enumerator kENET_QOS_BurstLen128

DMA burst length 128. eight times enabled.

enumerator kENET_QOS_BurstLen256

DMA burst length 256. eight times enabled.

enum _enet_qos_desc_flag

Define the flag for the descriptor.

Values:

enumerator kENET_QOS_MiddleFlag

It's a middle descriptor of the frame.

enumerator kENET_QOS_LastFlagOnly

It's the last descriptor of the frame.

enumerator kENET_QOS_FirstFlagOnly

It's the first descriptor of the frame.

enumerator kENET_QOS_FirstLastFlag

It's the first and last descriptor of the frame.

enum _enet_qos_systime_op

Define the system time adjust operation control.

Values:

enumerator kENET_QOS_SystimeAdd
System time add to.

enumerator kENET_QOS_SystimeSubtract
System time subtract.

enum _enet_qos_ts_rollover_type
Define the system time rollover control.

Values:

enumerator kENET_QOS_BinaryRollover
System time binary rollover.

enumerator kENET_QOS_DigitalRollover
System time digital rollover.

enum _enet_qos_special_config
Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, these 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_qos_config_t.

Note: "kENET_QOS_StoreAndForward" is recommended to be set.

Values:

enumerator kENET_QOS_DescDoubleBuffer
The double buffer is used in the tx/rx descriptor.

enumerator kENET_QOS_StoreAndForward
The rx/tx store and forward enable.

enumerator kENET_QOS_PromiscuousEnable
The promiscuous enabled.

enumerator kENET_QOS_FlowControlEnable
The flow control enabled.

enumerator kENET_QOS_BroadCastRxDisable
The broadcast disabled.

enumerator kENET_QOS_MulticastAllEnable
All multicast are passed.

enumerator kENET_QOS_8023AS2KPacket
8023as support for 2K packets.

enumerator kENET_QOS_HashMulticastEnable
The multicast packets are filtered through hash table.

enumerator kENET_QOS_RxChecksumOffloadEnable
The Rx checksum offload enabled.

enum _enet_qos_dma_interrupt_enable
List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-bot encoding to allow a logical OR of multiple members.

Values:

enumerator kENET_QOS_DmaTx
Tx interrupt.

enumerator kENET_QOS_DmaTxStop
Tx stop interrupt.

enumerator kENET_QOS_DmaTxBuffUnavail
Tx buffer unavailable.

enumerator kENET_QOS_DmaRx
Rx interrupt.

enumerator kENET_QOS_DmaRxBuffUnavail
Rx buffer unavailable.

enumerator kENET_QOS_DmaRxStop
Rx stop.

enumerator kENET_QOS_DmaRxWatchdogTimeout
Rx watchdog timeout.

enumerator kENET_QOS_DmaEarlyTx
Early transmit.

enumerator kENET_QOS_DmaEarlyRx
Early receive.

enumerator kENET_QOS_DmaBusErr
Fatal bus error.

enum _enet_qos_mac_interrupt_enable
List of mac interrupts supported by the ENET interrupt. This enumeration uses one-bot encoding to allow a logical OR of multiple members.

Values:

enumerator kENET_QOS_MacTimestamp

enum _enet_qos_event
Defines the common interrupt event for callback use.

Values:

enumerator kENET_QOS_RxIntEvent
Receive interrupt event.

enumerator kENET_QOS_TxIntEvent
Transmit interrupt event.

enumerator kENET_QOS_WakeUpIntEvent
Wake up interrupt event.

enumerator kENET_QOS_TimeStampIntEvent
Time stamp interrupt event.

enum _enet_qos_queue_mode
Define the MTL mode for multiple queues/rings.

Values:

enumerator kENET_QOS_AVB_Mode
Enable queue in AVB mode.

enumerator kENET_QOS_DCB_Mode
Enable queue in DCB mode.

enum _enet_qos_mtl_multiqueue_txsche
Define the MTL tx scheduling algorithm for multiple queues/rings.

Values:

enumerator kENET_QOS_txWeightRR
Tx weight round-robin.

enumerator kENET_QOS_txWeightFQ
Tx weight fair queuing.

enumerator kENET_QOS_txDefictWeightRR
Tx deficit weighted round-robin.

enumerator kENET_QOS_txStrPrio
Tx strict priority.

enum _enet_qos_mtl_multiqueue_rxsche
Define the MTL rx scheduling algorithm for multiple queues/rings.

Values:

enumerator kENET_QOS_rxStrPrio
Rx strict priority, Queue 0 has the lowest priority.

enumerator kENET_QOS_rxWeightStrPrio
Weighted Strict Priority.

enum _enet_qos_mtl_rxqueuemap
Define the MTL rx queue and DMA channel mapping.

Values:

enumerator kENET_QOS_StaticDirctMap
The received fame in rx Qn(n = 0,1) directly map to dma channel n.

enumerator kENET_QOS_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_qos_rx_queue_route
Defines the package type for receive queue routing.

Values:

enumerator kENET_QOS_PacketNoQ

enumerator kENET_QOS_PacketAVCPQ

enumerator kENET_QOS_PacketPTPQ

enumerator kENET_QOS_PacketUPQ

enumerator kENET_QOS_PacketMCBCQ

enum _enet_qos_ptp_event_type
Defines the ENET PTP message related constant.

Values:

enumerator kENET_QOS_PtpEventMsgType
PTP event message type.

enumerator kENET_QOS_PtpSrcPortIdLen
PTP message sequence id length.

enumerator kENET_QOS_PtpEventPort
PTP event port number.

enumerator kENET_QOS_PtpGnrlPort
PTP general port number.

enum _enet_qos_ptp_pps_instance
Defines the PPS instance numbers.

Values:

enumerator kENET_QOS_PtpPpsIstance0
PPS instance 0.

enumerator kENET_QOS_PtpPpsIstance1
PPS instance 1.

enumerator kENET_QOS_PtpPpsIstance2
PPS instance 2.

enumerator kENET_QOS_PtpPpsIstance3
PPS instance 3.

enum _enet_qos_ptp_pps_trgt_mode
Defines the Target Time register mode.

Values:

enumerator kENET_QOS_PtpPpsTrgtModeOnlyInt
Only interrupts.

enumerator kENET_QOS_PtpPpsTrgtModeIntSt
Both interrupt and output signal.

enumerator kENET_QOS_PtpPpsTrgtModeOnlySt
Only output signal.

enum _enet_qos_ptp_pps_cmd
Defines commands for ppscmd register.

Values:

enumerator kENET_QOS_PtpPpsCmdNC
No Command.

enumerator kENET_QOS_PtpPpsCmdSSP
Start Single Pulse.

enumerator kENET_QOS_PtpPpsCmdSPT
Start Pulse Train.

enumerator kENET_QOS_PtpPpsCmdCS
Cancel Start.

enumerator kENET_QOS_PtpPpsCmdSPTAT
Stop Pulse Train At Time.

enumerator kENET_QOS_PtpPpsCmdSPTI
Stop Pulse Train Immediately.

enumerator kENET_QOS_PtpPpsCmdCSPT
Cancel Stop Pulse Train.

enum _enet_qos_ets_list_length
Defines the enmueration of ETS list length.

Values:

enumerator kENET_QOS_Ets_List_64
List length of 64

enumerator kENET_QOS_Ets_List_128
List length of 128

enumerator kENET_QOS_Ets_List_256
List length of 256

enumerator kENET_QOS_Ets_List_512
List length of 512

enumerator kENET_QOS_Ets_List_1024
List length of 1024

enum _enet_qos_ets_gccr_addr
Defines the enmueration of ETS gate control address.

Values:

enumerator kENET_QOS_Ets_btr_low
BTR Low

enumerator kENET_QOS_Ets_btr_high
BTR High

enumerator kENET_QOS_Ets_ctr_low
CTR Low

enumerator kENET_QOS_Ets_ctr_high
CTR High

enumerator kENET_QOS_Ets_ter
TER

enumerator kENET_QOS_Ets_llr
LLR

enum _enet_qos_rxp_dma_chn
Defines the enmueration of DMA channel used for rx parser entry.

Values:

enumerator kENET_QOS_Rxp_DMAChn0
DMA Channel 0 used for RXP entry match

enumerator kENET_QOS_Rxp_DMAChn1
DMA Channel 1 used for RXP entry match

enumerator kENET_QOS_Rxp_DMAChn2
DMA Channel 2 used for RXP entry match

enumerator kENET_QOS_Rxp_DMAChn3
DMA Channel 3 used for RXP entry match

enumerator kENET_QOS_Rxp_DMAChn4
DMA Channel 4 used for RXP entry match

enum _enet_qos_tx_offload
Define the Tx checksum offload options.

Values:

enumerator kENET_QOS_TxOffloadDisable
Disable Tx checksum offload.

enumerator kENET_QOS_TxOffloadIPHeader
Enable IP header checksum calculation and insertion.

enumerator kENET_QOS_TxOffloadIPHeaderPlusPayload
Enable IP header and payload checksum calculation and insertion.

enumerator kENET_QOS_TxOffloadAll
Enable IP header, payload and pseudo header checksum calculation and insertion.

typedef enum *_enet_qos_mii_mode* enet_qos_mii_mode_t
Defines the MII/RGMII mode for data interface between the MAC and the PHY.

typedef enum *_enet_qos_mii_speed* enet_qos_mii_speed_t
Defines the 10/100/1000 Mbps speed for the MII data interface.

typedef enum *_enet_qos_mii_duplex* enet_qos_mii_duplex_t
Defines the half or full duplex for the MII data interface.

typedef enum *_enet_qos_mii_normal_opcode* enet_qos_mii_normal_opcode
Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

typedef enum *_enet_qos_dma_burstlen* enet_qos_dma_burstlen
Define the DMA maximum transmit burst length.

typedef enum *_enet_qos_desc_flag* enet_qos_desc_flag
Define the flag for the descriptor.

typedef enum *_enet_qos_systime_op* enet_qos_systime_op
Define the system time adjust operation control.

typedef enum *_enet_qos_ts_rollover_type* enet_qos_ts_rollover_type
Define the system time rollover control.

typedef enum *_enet_qos_special_config* enet_qos_special_config_t
Defines some special configuration for ENET.

These control flags are provided for special user requirements. Normally, these 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_qos_config_t.

Note: "kENET_QOS_StoreAndForward" is recommended to be set.

---

typedef enum *_enet_qos_dma_interrupt_enable* enet_qos_dma_interrupt_enable_t
List of DMA interrupts supported by the ENET interrupt. This enumeration uses one-bot encoding to allow a logical OR of multiple members.

typedef enum *_enet_qos_mac_interrupt_enable* enet_qos_mac_interrupt_enable_t
List of mac interrupts supported by the ENET interrupt. This enumeration uses one-bot encoding to allow a logical OR of multiple members.

typedef enum *_enet_qos_event* enet_qos_event_t

Defines the common interrupt event for callback use.

typedef enum *_enet_qos_queue_mode* enet_qos_queue_mode_t

Define the MTL mode for multiple queues/rings.

typedef enum *_enet_qos_mtl_multiqueue_txsche* enet_qos_mtl_multiqueue_txsche

Define the MTL tx scheduling algorithm for multiple queues/rings.

typedef enum *_enet_qos_mtl_multiqueue_rxsche* enet_qos_mtl_multiqueue_rxsche

Define the MTL rx scheduling algorithm for multiple queues/rings.

typedef enum *_enet_qos_mtl_rxqueuemap* enet_qos_mtl_rxqueuemap_t

Define the MTL rx queue and DMA channel mapping.

typedef enum *_enet_qos_rx_queue_route* enet_qos_rx_queue_route_t

Defines the package type for receive queue routing.

typedef enum *_enet_qos_ptp_event_type* enet_qos_ptp_event_type_t

Defines the ENET PTP message related constant.

typedef enum *_enet_qos_ptp_pps_instance* enet_qos_ptp_pps_instance_t

Defines the PPS instance numbers.

typedef enum *_enet_qos_ptp_pps_trgt_mode* enet_qos_ptp_pps_trgt_mode_t

Defines the Target Time register mode.

typedef enum *_enet_qos_ptp_pps_cmd* enet_qos_ptp_pps_cmd_t

Defines commands for ppscmd register.

typedef enum *_enet_qos_ets_list_length* enet_qos_ets_list_length_t

Defines the enmueration of ETS list length.

typedef enum *_enet_qos_ets_gccr_addr* enet_qos_ets_gccr_addr_t

Defines the enmueration of ETS gate control address.

typedef enum *_enet_qos_rxp_dma_chn* enet_qos_rxp_dma_chn_t

Defines the enmueration of DMA channel used for rx parser entry.

typedef enum *_enet_qos_tx_offload* enet_qos_tx_offload_t

Define the Tx checksum offload options.

typedef struct *_enet_qos_rx_bd_struct* enet_qos_rx_bd_struct_t

Defines the receive descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the receive descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the receive data process.

typedef struct *_enet_qos_tx_bd_struct* enet_qos_tx_bd_struct_t

Defines the transmit descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the transmit descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the transmit data process.

typedef struct *_enet_qos_tx_bd_config_struct* enet_qos_tx_bd_config_struct_t

Defines the Tx BD configuration structure.

typedef struct *_enet_qos_ptp_time* enet_qos_ptp_time_t

Defines the ENET PTP time stamp structure.

typedef struct *enet_qos_frame_info* enet_qos_frame_info_t

Defines the frame info structure.

typedef struct *_enet_qos_tx_dirty_ring* enet_qos_tx_dirty_ring_t

Defines the ENET transmit dirty addresses ring/queue structure.

typedef struct *_enet_qos_ptp_config* enet_qos_ptp_config_t

Defines the ENET PTP configuration structure.

typedef struct *_enet_qos_est_gate_op* enet_qos_est_gate_op_t

Defines the EST gate operation structure.

typedef struct *_enet_qos_est_gcl* enet_qos_est_gcl_t

Defines the EST gate control list structure.

typedef struct *_enet_qos_rxp_config* enet_qos_rxp_config_t

Defines the ENET_QOS Rx parser configuration structure.

typedef struct *_enet_qos_buffer_config* enet_qos_buffer_config_t

Defines the buffer descriptor configure structure.

Note:

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 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 memories 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.

---

typedef struct *_enet_qos_cbs_config* enet_qos_cbs_config_t

Defines the CBS configuration for queue.

typedef struct *enet_qos_tx_queue_config* enet_qos_queue_tx_config_t

Defines the queue configuration structure.

typedef struct *enet_qos_rx_queue_config* enet_qos_queue_rx_config_t

Defines the queue configuration structure.

typedef struct *enet_qos_multiqueue_config* enet_qos_multiqueue_config_t

Defines the configuration when multi-queue is used.

typedef void *(*enet_qos_rx_alloc_callback_t)(ENET_QOS_Type *base, void *userData, uint8_t channel)

Defines the Rx memory buffer alloc function pointer.

typedef void (*enet_qos_rx_free_callback_t)(ENET_QOS_Type *base, void *buffer, void *userData, uint8_t channel)

Defines the Rx memory buffer free function pointer.

typedef struct *_enet_qos_config* enet_qos_config_t

Defines the basic configuration structure for the ENET device.

Note: 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_qos_handle* enet_qos_handle_t

typedef void (*enet_qos_callback_t)(ENET_QOS_Type *base, *enet_qos_handle_t* *handle, *enet_qos_event_t* event, uint8_t channel, void *userData)

ENET callback function.

typedef struct *_enet_qos_tx_bd_ring* enet_qos_tx_bd_ring_t

Defines the ENET transmit buffer descriptor ring/queue structure.

typedef struct *_enet_qos_rx_bd_ring* enet_qos_rx_bd_ring_t

Defines the ENET receive buffer descriptor ring/queue structure.

typedef struct *_enet_qos_state* enet_qos_state_t

Defines the ENET state structure.

Note: The structure contains saved state for the instance. It could be stored in enet_qos_handle_t, but that's used only with the transactional API.

---

typedef struct *_enet_qos_buffer_struct* enet_qos_buffer_struct_t

Defines the frame buffer structure.

typedef struct *_enet_qos_rx_frame_error* enet_qos_rx_frame_error_t

Defines the Rx frame error structure.

typedef struct *_enet_qos_rx_frame_attribute_struct* enet_qos_rx_frame_attribute_t

typedef struct *_enet_qos_rx_frame_struct* enet_qos_rx_frame_struct_t

Defines the Rx frame data structure.

typedef struct *_enet_qos_transfer_stats* enet_qos_transfer_stats_t

Defines the ENET QOS transfer statistics structure.

typedef void (*enet_qos_isr_t)(ENET_QOS_Type *base, *enet_qos_handle_t* *handle)

const *clock_ip_name_t* s_enetqosClock[]

Pointers to enet clocks for each instance.

void ENET_QOS_SetSYSControl(*enet_qos_mii_mode_t* miiMode)

Set ENET system configuration.

Note: User needs to provide the implementation because the implementation is SoC specific. This function set the phy selection and enable clock. It should be called before any other ethernet operation.

Parameters

- miiMode – The MII/RGMII/RMII mode for interface between the phy and Ethernet.

void ENET_QOS_EnableClock(bool enable)

Enable/Disable ENET qos clock.

Note: User needs to provide the implementation because the implementation is SoC specific. This function should be called before config RMII mode.

struct _enet_qos_rx_bd_struct

#include <fsl_enet_qos.h> Defines the receive descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the receive descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the receive data process.

Public Members

___IO uint32_t buff1Addr

Buffer 1 address

___IO uint32_t reserved

Reserved

___IO uint32_t buff2Addr

Buffer 2 or next descriptor address

___IO uint32_t control

Buffer 1/2 byte counts and control

struct _enet_qos_tx_bd_struct

#include <fsl_enet_qos.h> Defines the transmit descriptor structure has the read-format and write-back format structure. They both has the same size with different region definition. so we define the read-format region as the transmit descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the transmit data process.

Public Members

___IO uint32_t buff1Addr

Buffer 1 address

___IO uint32_t buff2Addr

Buffer 2 address

___IO uint32_t buffLen

Buffer 1/2 byte counts

___IO uint32_t controlStat

TDES control and status word

struct _enet_qos_tx_bd_config_struct

#include <fsl_enet_qos.h> Defines the Tx BD configuration structure.

Public Members

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_qos_tx_offload_t txOffloadOps
: The Tx checksum offload option.

enet_qos_desc_flag flag
: The flag of this tx desciriptor, see “enet_qos_desc_flag”.

struct __enet_qos_ptp_time
: *#include <fsl_enet_qos.h>* Defines the ENET PTP time stamp structure.

Public Members

uint64_t second
: Second.

uint32_t nanosecond
: Nanosecond.

struct enet_qos_frame_info
: *#include <fsl_enet_qos.h>* Defines the frame info structure.

Public Members

void *context
: User specified data, could be buffer address for free

bool isTsAvail
: Flag indicates timestamp available status

enet_qos_ptp_time_t timeStamp
: Timestamp of frame

struct __enet_qos_tx_dirty_ring
: *#include <fsl_enet_qos.h>* Defines the ENET transmit dirty addresses ring/queue structure.

Public Members

enet_qos_frame_info_t *txDirtyBase
: Dirty buffer descriptor base address pointer.

uint16_t txGenIdx
: tx generate index.

uint16_t txConsumIdx
: tx consume index.

uint16_t txRingLen
: tx ring length.

bool isFull
: tx ring is full flag, add this parameter to avoid waste one element.

struct _enet_qos_ptp_config
: *#include <fsl_enet_qos.h>* Defines the ENET PTP configuration structure.

Public Members

bool fineUpdateEnable
: Use the fine update.

uint32_t defaultAddend
: Default addend value when fine update is enable, could be 2^32 / (refClk_Hz / ENET_QOS_MICRSECS_ONESECOND / ENET_QOS_SYSTIME_REQUIRED_CLK_MHZ).

bool ptp1588V2Enable
: The desired system time frequency. Must be lower than reference clock. (Only used with fine correction method). ptp 1588 version 2 is used.

enet_qos_ts_rollover_type tsRollover
: 1588 time nanosecond rollover.

struct _enet_qos_est_gate_op
: *#include <fsl_enet_qos.h>* Defines the EST gate operation structure.

struct _enet_qos_est_gcl
: *#include <fsl_enet_qos.h>* Defines the EST gate control list structure.

Public Members

bool enable
: Enable or disable EST

uint64_t cycleTime
: Base Time 32 bits seconds 32 bits nanoseconds

uint32_t extTime
: Cycle Time 32 bits seconds 32 bits nanoseconds

uint32_t numEntries
: Time Extension 32 bits seconds 32 bits nanoseconds

enet_qos_est_gate_op_t *opList
: Number of entries

struct _enet_qos_rxp_config
: *#include <fsl_enet_qos.h>* Defines the ENET_QOS Rx parser configuration structure.

Public Members

uint32_t matchEnable
: 4-byte match data used for comparing with incoming packet

uint8_t acceptFrame
When matchEnable is set to 1, the matchData is used for comparing

uint8_t rejectFrame
When acceptFrame = 1 and data is matched, the frame will be sent to DMA channel

uint8_t inverseMatch
When rejectFrame = 1 and data is matched, the frame will be dropped

uint8_t nextControl
Inverse match

uint8_t reserved
Next instruction indexing control

uint8_t frameOffset
Reserved control fields

uint8_t okIndex
Frame offset in the packet data to be compared for match, in terms of 4 bytes.

uint8_t dmaChannel
Memory Index to be used next.

uint32_t reserved2
The DMA channel enet_qos_rxp_dma_chn_t used for receiving the frame when frame match and acceptFrame = 1

struct __enet_qos_buffer_config
#include <fsl_enet_qos.h> Defines the buffer descriptor configure structure.

Note:

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 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 memories 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.

Public Members

uint8_t rxRingLen
The length of receive buffer descriptor ring.

uint8_t txRingLen
The length of transmit buffer descriptor ring.

enet_qos_tx_bd_struct_t *txDescStartAddrAlign
Aligned transmit descriptor start address.

enet_qos_tx_bd_struct_t *txDescTailAddrAlign
Aligned transmit descriptor tail address.

enet_qos_frame_info_t *txDirtyStartAddr
Start address of the dirty tx frame information.

enet_qos_rx_bd_struct_t *rxDescStartAddrAlign
Aligned receive descriptor start address.

enet_qos_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.

bool rxBuffNeedMaintain
Whether receive data buffer need cache maintain.

struct __enet_qos_cbs_config
#include <fsl_enet_qos.h> Defines the CBS configuration for queue.

Public Members

uint16_t sendSlope
Send slope configuration.

uint16_t idleSlope
Idle slope configuration.

uint32_t highCredit
High credit.

uint32_t lowCredit
Low credit.

struct enet_qos_tx_queue_config
#include <fsl_enet_qos.h> Defines the queue configuration structure.

Public Members

enet_qos_queue_mode_t mode
tx queue mode configuration.

uint32_t weight
Refer to the MTL TxQ Quantum Weight register.

uint32_t priority
Refer to Transmit Queue Priority Mapping register.

enet_qos_cbs_config_t *cbsConfig
CBS configuration if queue use AVB mode.

struct enet_qos_rx_queue_config
#include <fsl_enet_qos.h> Defines the queue configuration structure.

Public Members

enet_qos_queue_mode_t mode
rx queue mode configuration.

uint8_t mapChannel
tx queue map dma channel.

uint32_t priority
Rx queue priority.

enet_qos_rx_queue_route_t packetRoute
Receive packet routing.

struct enet_qos_multiqueue_config
#include <fsl_enet_qos.h> Defines the configuration when multi-queue is used.

Public Members

enet_qos_dma_burstlen burstLen
Burst len for the multi-queue.

uint8_t txQueueUse
Used Tx queue count.

enet_qos_mtl_multiqueue_txsche mtltxSche
Transmit schedule for multi-queue.

enet_qos_queue_tx_config_t txQueueConfig[ENET_QOS_DMA_CH_COUNT]
Tx Queue configuration.

uint8_t rxQueueUse
Used Rx queue count.

enet_qos_mtl_multiqueue_rxsche mtlrxSche
Receive schedule for multi-queue.

enet_qos_queue_rx_config_t rxQueueConfig[ENET_QOS_DMA_CH_COUNT]
Rx Queue configuration.

struct _enet_qos_config
#include <fsl_enet_qos.h> Defines the basic configuration structure for the ENET device.

Note: 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.

Public Members

uint16_t specialControl
The logic or of enet_qos_special_config_t

enet_qos_multiqueue_config_t *multiqueueCfg
Use multi-queue.

enet_qos_mii_mode_t miiMode
MII mode.

enet_qos_mii_speed_t miiSpeed
MII Speed.

enet_qos_mii_duplex_t miiDuplex
MII duplex.

uint16_t pauseDuration
Used in the tx flow control frame, only valid when kENET_QOS_FlowControlEnable is set.

enet_qos_ptp_config_t *ptpConfig
PTP 1588 feature configuration

uint32_t csrClock_Hz
CSR clock frequency in HZ.

enet_qos_rx_alloc_callback_t rxBuffAlloc
Callback to alloc memory, must be provided for zero-copy Rx.

enet_qos_rx_free_callback_t rxBuffFree
Callback to free memory, must be provided for zero-copy Rx.

struct __enet_qos_tx_bd_ring
#include <fsl_enet_qos.h> Defines the ENET transmit buffer descriptor ring/queue structure.

Public Members

enet_qos_tx_bd_struct_t *txBdBase
Buffer descriptor base address pointer.

uint16_t txGenIdx
tx generate index.

uint16_t txConsumIdx
tx consume index.

volatile uint16_t txDescUsed
tx descriptor used number.

uint16_t txRingLen
tx ring length.

struct __enet_qos_rx_bd_ring
#include <fsl_enet_qos.h> Defines the ENET receive buffer descriptor ring/queue structure.

Public Members

enet_qos_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.

struct __enet_qos_handle
#include <fsl_enet_qos.h> Defines the ENET handler structure.

Public Members

uint8_t txQueueUse
: Used tx queue count.

uint8_t rxQueueUse
: Used rx queue count.

bool doubleBuffEnable
: The double buffer is used in the descriptor.

bool rxintEnable
: Rx interrupt enabled.

bool rxMaintainEnable[ENET_QOS_DMA_CH_COUNT]
: Rx buffer cache maintain enabled.

enet_qos_rx_bd_ring_t rxBdRing[ENET_QOS_DMA_CH_COUNT]
: Receive buffer descriptor.

enet_qos_tx_bd_ring_t txBdRing[ENET_QOS_DMA_CH_COUNT]
: Transmit buffer descriptor.

enet_qos_tx_dirty_ring_t txDirtyRing[ENET_QOS_DMA_CH_COUNT]
: Transmit dirty buffers addresses.

uint32_t *rxBufferStartAddr[ENET_QOS_DMA_CH_COUNT]
: Rx buffer start address for reInitialize.

enet_qos_callback_t callback
: Callback function.

void *userData
: Callback function parameter.

uint8_t multicastCount[64]
: Multicast collisions counter

enet_qos_rx_alloc_callback_t rxBuffAlloc
: Callback to alloc memory, must be provided for zero-copy Rx.

enet_qos_rx_free_callback_t rxBuffFree
: Callback to free memory, must be provided for zero-copy Rx.

struct __enet_qos_state
: *#include <fsl_enet_qos.h>* Defines the ENET state structure.

Note: The structure contains saved state for the instance. It could be stored in enet_qos_handle_t, but that's used only with the transactional API.

Public Members

enet_qos_mii_mode_t miiMode
: MII mode.

struct __enet_qos_buffer_struct
: *#include <fsl_enet_qos.h>* Defines the frame buffer structure.

Public Members

void *buffer
The buffer store the whole or partial frame.

uint16_t length
The byte length of this buffer.

struct __enet_qos_rx_frame_error
#include <fsl_enet_qos.h> Defines the Rx frame error structure.

Public Members

bool rxDstAddrFilterErr
Destination Address Filter Fail.

bool rxSrcAddrFilterErr
SA Address Filter Fail.

bool rxDribbleErr
Dribble error.

bool rxReceiveErr
Receive error.

bool rxOverFlowErr
Receive over flow.

bool rxWatchDogErr
Watch dog timeout.

bool rxGaintPacketErr
Receive gaint packet.

bool rxCrcErr
Receive CRC error.

struct __enet_qos_rx_frame_attribute_struct
#include <fsl_enet_qos.h>

Public Members

bool isTsAvail
Rx frame timestamp is available or not.

enet_qos_ptp_time_t timestamp
The nanosecond part timestamp of this Rx frame.

struct __enet_qos_rx_frame_struct
#include <fsl_enet_qos.h> Defines the Rx frame data structure.

Public Members

enet_qos_buffer_struct_t *rxBuffArray
Rx frame buffer structure.

uint16_t totLen
Rx frame total length.

enet_qos_rx_frame_attribute_t rxAttribute
Rx frame attribute structure.

enet_qos_rx_frame_error_t rxFrameError
Rx frame error.

struct __enet_qos_transfer_stats
#include <fsl_enet_qos.h> Defines the ENET QOS transfer statistics structure.

Public Members

uint32_t statsRxFrameCount
Rx frame number.

uint32_t statsRxCrcErr
Rx frame number with CRC error.

uint32_t statsRxAlignErr
Rx frame number with alignment error.

uint32_t statsRxLengthErr
Rx frame length field doesn't equal to packet size.

uint32_t statsRxFifoOverflowErr
Rx FIFO overflow count.

uint32_t statsTxFrameCount
Tx frame number.

uint32_t statsTxFifoUnderRunErr
Tx FIFO underrun count.

2.17 FlexCAN: Flex Controller Area Network Driver

2.18 FlexCAN Driver

bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)
Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN_FDInit to enable CAN FD mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)

Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080. For 32 bytes payload frame, the second mailbox is ((1/12)*512 + 1%12*40)/4 = 10, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – Mailbox index.

Returns

Mailbox address offset in word.

status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns

kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

uint32_t FLEXCAN_GetInstance(CAN_Type *base)

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN instance.

bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.

- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

void FLEXCAN_Init(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                 = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate                = 1000000U;
flexcanConfig.maxMbNum               = 16;
flexcanConfig.enableLoopBack         = false;
flexcanConfig.enableSelfWakeup       = false;
flexcanConfig.enableIndividMask      = false;
flexcanConfig.enableDoze             = false;
flexcanConfig.disableSelfReception   = false;
flexcanConfig.enableListenOnlyMode   = false;
flexcanConfig.timingConfig           = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 40000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t bitRateFD, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.
- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

void FLEXCAN_FDInit(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz, *flexcan_mb_size_t* dataSize, bool brs)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                 = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate                = 1000000U;
flexcanConfig.bitRateFD              = 2000000U;
flexcanConfig.maxMbNum               = 16;
flexcanConfig.enableLoopBack         = false;
flexcanConfig.enableSelfWakeup       = false;
flexcanConfig.enableIndividMask      = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze             = false;
flexcanConfig.timingConfig           = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 80000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

void FLEXCAN_Deinit(CAN_Type *base)

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN_GetDefaultConfig(*flexcan_config_t* *pConfig)

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enablePretendedeNetworking = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian; flexcanConfig.timingConfig = timingConfig;

Parameters

- pConfig – Pointer to the FlexCAN configuration structure.

void FLEXCAN_SetTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetBitRate() instead.

Note: Calling FLEXCAN_SetTimingConfig() overrides the bit rate set in FLEXCAN_Init() or FLEXCAN_SetBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on FLEXCAN_CalculateImprovedTimingValues() API calculated timing values.

Note: Calling FLEXCAN_SetBitRate() overrides the bit rate set in FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRate_Bps – Bit rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only Nominal phase) successfully.

void FLEXCAN_SetFDTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_SetFDBitRate() to set both Nominal/Data bit Rate instead.

Note: Calling FLEXCAN_SetFDTimingConfig() overrides the data phase bit rate set in FLEXCAN_FDInit()/FLEXCAN_SetFDBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRateN_Bps, uint32_t bitRateD_Bps)

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on FLEXCAN_FDCalculateImprovedTimingValues() API calculated timing values.

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRateN_Bps – Nominal bit Rate in Bps.

- bitRateD_Bps – Data bit Rate in Bps.

Returns

kStatus_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the FLEXCAN_Init().

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the FLEXCAN_Init(). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN_SetTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbEmpty, kFLEXCAN_RxMbFull or kFLEXCAN_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.

- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pFrame)

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const *flexcan_rx_fifo_config_t* *pRxFifoConfig, bool enable)

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note: Legacy Rx FIFO only can receive classic CAN message.

Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Legacy Rx FIFO.
  - true: Enable Legacy Rx FIFO.
  - false: Disable Legacy Rx FIFO.

void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const *flexcan_enhanced_rx_fifo_config_t* *pConfig, bool enable)

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

Note: Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
  - true: Enable Enhanced Rx FIFO.
  - false: Disable Enhanced Rx FIFO.

void FLEXCAN_SetPNConfig(CAN_Type *base, const *flexcan_pn_config_t* *pConfig)

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators _flexcan_flags. To check the specific status, compare the return value with enumerators in _flexcan_flags.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the _flexcan_flags.

static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of _flexcan_flags.

static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base,
flexcan_memory_error_report_status_t *errorStatus)

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

Parameters

- base – FlexCAN peripheral base address.

Returns
The number of received wake up msessages.

static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.

Returns
The number of available CAN messages stored in the Enhanced Rx FIFO.

static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of _flexcan_interrupt_enable.

static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see _flexcan_interrupt_enable.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to disable. Logical OR of _flexcan_interrupt_enable.

static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.

- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- enable – true to enable, false to disable.

static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN Rx FIFO Head address.

static inline *status_t* FLEXCAN_Enable(CAN_Type *base, bool enable)

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

Parameters

- base – FlexCAN base pointer.
- enable – true to enable, false to disable.

Returns

kStatus_Success Enable FlexCAN module successful kStatus_Timeout Timeout when wait for Low-Power Mode Acknowledge

status_t FLEXCAN_WriteTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pTxFrame)

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.

- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.

status_t FLEXCAN_WriteFDTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_fd_frame_t* *pTxFrame)

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadFDRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.

status_t FLEXCAN_ReadRxFifo(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadEnhancedRxFifo(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadPNWakeUpMB(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.
- mbIdx – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

Return values

- kStatus_Success – - Read Message from Wake up Message Buffer successfully.
- kStatus_Fail – - Wake up Message Buffer has no valid content.

status_t FLEXCAN_TransferFDSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferFDReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferFDSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferFDReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.

- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

void FLEXCAN_TransferFDAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

void FLEXCAN_TransferFDAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

status_t FLEXCAN_TransferSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveFifoBlocking(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveEnhancedFifoBlocking(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

void FLEXCAN_TransferCreateHandle(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_transfer_callback_t* callback, void *userData)

Initializes the FlexCAN handle.

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t FLEXCAN_TransferSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the flexcan_fifo_transfer_t.

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

status_t FLEXCAN_TransferGetReceiveFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

status_t FLEXCAN_TransferReceiveEnhancedFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the ref flexcan_fifo_transfer_t.@

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

uint32_t FLEXCAN_GetTimeStamp(*flexcan_handle_t* *handle, uint8_t mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback

function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

static inline uint32_t FLEXCAN_GetHighResolutionTimeStamp(CAN_Type *base, uint8_t mbIdx)

void FLEXCAN_TransferAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferAbortReceiveEnhancedFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Enhanced Rx FIFO process.

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MbHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle, uint32_t startMbIdx, uint32_t endMbIdx)

FlexCAN Message Buffer IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

void FLEXCAN_EhancedRxFifoHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Ehanced Rx FIFO IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_BusoffErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Bus Off, Error and Warning IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_PNWakeUpHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Pretended Networking Wake-up IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MemoryErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Memory Error IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

FSL_FLEXCAN_DRIVER_VERSION

FlexCAN driver version.

FlexCAN transfer status.

Values:

enumerator kStatus_FLEXCAN_TxBusy

Tx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_TxIdle

Tx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_TxSwitchToRx

Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus_FLEXCAN_RxBusy
Rx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_RxIdle
Rx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_RxOverflow
Rx Message Buffer is Overflowed.

enumerator kStatus_FLEXCAN_RxFifoBusy
Rx Message FIFO is Busy.

enumerator kStatus_FLEXCAN_RxFifoIdle
Rx Message FIFO is Idle.

enumerator kStatus_FLEXCAN_RxFifoOverflow
Rx Message FIFO is overflowed.

enumerator kStatus_FLEXCAN_RxFifoWarning
Rx Message FIFO is almost overflowed.

enumerator kStatus_FLEXCAN_RxFifoDisabled
Rx Message FIFO is disabled during reading.

enumerator kStatus_FLEXCAN_ErrorStatus
FlexCAN Module Error and Status.

enumerator kStatus_FLEXCAN_WakeUp
FlexCAN is waken up from STOP mode.

enumerator kStatus_FLEXCAN_UnHandled
UnHadled Interrupt asserted.

enumerator kStatus_FLEXCAN_RxRemote
Rx Remote Message Received in Mail box.

enumerator kStatus_FLEXCAN_RxFifoUnderflow
Enhanced Rx Message FIFO is underflow.

enumerator kStatus_FLEXCAN_MemoryError
FlexCAN Memory Error.

enum _flexcan_frame_format
FlexCAN frame format.

Values:

enumerator kFLEXCAN_FrameFormatStandard
Standard frame format attribute.

enumerator kFLEXCAN_FrameFormatExtend
Extend frame format attribute.

enum _flexcan_frame_type
FlexCAN frame type.

Values:

enumerator kFLEXCAN_FrameTypeData
Data frame type attribute.

enumerator kFLEXCAN_FrameTypeRemote
Remote frame type attribute.

enum _flexcan_clock_source

FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

Values:

enumerator kFLEXCAN_ClkSrcOsc

FlexCAN Protocol Engine clock from Oscillator.

enumerator kFLEXCAN_ClkSrcPeri

FlexCAN Protocol Engine clock from Peripheral Clock.

enumerator kFLEXCAN_ClkSrc0

FlexCAN Protocol Engine clock selected by user as SRC == 0.

enumerator kFLEXCAN_ClkSrc1

FlexCAN Protocol Engine clock selected by user as SRC == 1.

enum _flexcan_wake_up_source

FlexCAN wake up source.

Values:

enumerator kFLEXCAN_WakeupSrcUnfiltered

FlexCAN uses unfiltered Rx input to detect edge.

enumerator kFLEXCAN_WakeupSrcFiltered

FlexCAN uses filtered Rx input to detect edge.

enum _flexcan_endianness

FlexCAN payload endianness.

Values:

enumerator kFLEXCAN_bigEndian

Transmit frame with MSB first, receive frame with big-endian format.

enumerator kFLEXCAN_littleEndian

Transmit frame with LSB first, receive frame with little-endian format.

enum _flexcan_MB_timestamp_base

FlexCAN timebase used for capturing 16-bit TIME_STAMP field of message buffer.

Values:

enumerator kFLEXCAN_CANTimer

FlexCAN free-running timer.

enumerator kFLEXCAN_Lower16bitsHRTimer

Lower 16 bits of high-resolution on-chip timer.

enumerator kFLEXCAN_Upper16bitsHRTimer

Upper 16 bits of high-resolution on-chip timer.

enum _flexcan_capture_point

FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

Values:

enumerator kFLEXCAN_CANFrameID2ndBit

Second bit of identifier field of any frame is on the CAN bus. HR_TIME_STAMPn register will not capture 32-bit counter value.

enumerator kFLEXCAN_CANFrameEnd

End of the CAN frame.

enumerator kFLEXCAN_CANFrameStart

Start of the CAN frame.

enumerator kFLEXCAN_CANFDFrameRes

Start of frame for classical CAN frames; res bit for CAN FD frames.

enum _flexcan_rx_fifo_filter_type

FlexCAN Rx Fifo Filter type.

Values:

enumerator kFLEXCAN_RxFifoFilterTypeA

One full ID (standard and extended) per ID Filter element.

enumerator kFLEXCAN_RxFifoFilterTypeB

Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeC

Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeD

All frames rejected.

enum _flexcan_mb_size

FlexCAN Message Buffer Payload size.

Values:

enumerator kFLEXCAN_8BperMB

Selects 8 bytes per Message Buffer.

enumerator kFLEXCAN_16BperMB

Selects 16 bytes per Message Buffer.

enumerator kFLEXCAN_32BperMB

Selects 32 bytes per Message Buffer.

enumerator kFLEXCAN_64BperMB

Selects 64 bytes per Message Buffer.

enum _flexcan_fd_frame_length

FlexCAN CAN FD frame supporting data length (available DLC values).

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

Values:

enumerator kFLEXCAN_0BperFrame

Frame contains 0 valid data bytes.

enumerator kFLEXCAN_1BperFrame

Frame contains 1 valid data bytes.

enumerator kFLEXCAN_2BperFrame
Frame contains 2 valid data bytes.

enumerator kFLEXCAN_3BperFrame
Frame contains 3 valid data bytes.

enumerator kFLEXCAN_4BperFrame
Frame contains 4 valid data bytes.

enumerator kFLEXCAN_5BperFrame
Frame contains 5 valid data bytes.

enumerator kFLEXCAN_6BperFrame
Frame contains 6 valid data bytes.

enumerator kFLEXCAN_7BperFrame
Frame contains 7 valid data bytes.

enumerator kFLEXCAN_8BperFrame
Frame contains 8 valid data bytes.

enumerator kFLEXCAN_12BperFrame
Frame contains 12 valid data bytes.

enumerator kFLEXCAN_16BperFrame
Frame contains 16 valid data bytes.

enumerator kFLEXCAN_20BperFrame
Frame contains 20 valid data bytes.

enumerator kFLEXCAN_24BperFrame
Frame contains 24 valid data bytes.

enumerator kFLEXCAN_32BperFrame
Frame contains 32 valid data bytes.

enumerator kFLEXCAN_48BperFrame
Frame contains 48 valid data bytes.

enumerator kFLEXCAN_64BperFrame
Frame contains 64 valid data bytes.

enum _flexcan_efifo_dma_per_read_length
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

Values:

enumerator kFLEXCAN_1WordPerRead
Transfer 1 32-bit words (CS).

enumerator kFLEXCAN_2WordPerRead
Transfer 2 32-bit words (CS + ID).

enumerator kFLEXCAN_3WordPerRead
Transfer 3 32-bit words (CS + ID + 1~4 bytes data).

enumerator kFLEXCAN_4WordPerRead
Transfer 4 32-bit words (CS + ID + 5~8 bytes data).

enumerator kFLEXCAN_5WordPerRead
Transfer 5 32-bit words (CS + ID + 9~12 bytes data).

enumerator kFLEXCAN_6WordPerRead
Transfer 6 32-bit words (CS + ID + 13~16 bytes data).

enumerator kFLEXCAN_7WordPerRead
Transfer 7 32-bit words (CS + ID + 17~20 bytes data).

enumerator kFLEXCAN_8WordPerRead
Transfer 8 32-bit words (CS + ID + 21~24 bytes data).

enumerator kFLEXCAN_9WordPerRead
Transfer 9 32-bit words (CS + ID + 25~28 bytes data).

enumerator kFLEXCAN_10WordPerRead
Transfer 10 32-bit words (CS + ID + 29~32 bytes data).

enumerator kFLEXCAN_11WordPerRead
Transfer 11 32-bit words (CS + ID + 33~36 bytes data).

enumerator kFLEXCAN_12WordPerRead
Transfer 12 32-bit words (CS + ID + 37~40 bytes data).

enumerator kFLEXCAN_13WordPerRead
Transfer 13 32-bit words (CS + ID + 41~44 bytes data).

enumerator kFLEXCAN_14WordPerRead
Transfer 14 32-bit words (CS + ID + 45~48 bytes data).

enumerator kFLEXCAN_15WordPerRead
Transfer 15 32-bit words (CS + ID + 49~52 bytes data).

enumerator kFLEXCAN_16WordPerRead
Transfer 16 32-bit words (CS + ID + 53~56 bytes data).

enumerator kFLEXCAN_17WordPerRead
Transfer 17 32-bit words (CS + ID + 57~60 bytes data).

enumerator kFLEXCAN_18WordPerRead
Transfer 18 32-bit words (CS + ID + 61~64 bytes data).

enumerator kFLEXCAN_19WordPerRead
Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT).

enumerator kFLEXCAN_20WordPerRead
Transfer 20 32-bit words (CS + ID + 64 bytes data + ID HIT + HR timestamp).

enum _flexcan_rx_fifo_priority
FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

Values:

enumerator kFLEXCAN_RxFifoPrioLow
Matching process start from Rx Message Buffer first.

enumerator kFLEXCAN_RxFifoPrioHigh
Matching process start from Enhanced/Legacy Rx FIFO first.

enum _flexcan_interrupt_enable

FlexCAN interrupt enable enumerations.

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

Note: FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Values:

enumerator kFLEXCAN_BusOffInterruptEnable

Bus Off interrupt, use bit 15.

enumerator kFLEXCAN_ErrorInterruptEnable

CAN Error interrupt, use bit 14.

enumerator kFLEXCAN_TxWarningInterruptEnable

Tx Warning interrupt, use bit 11.

enumerator kFLEXCAN_RxWarningInterruptEnable

Rx Warning interrupt, use bit 10.

enumerator kFLEXCAN_FDErrorInterruptEnable

CAN FD Error interrupt, use bit 31.

enumerator kFLEXCAN_PNMatchWakeUpInterruptEnable

PN Match Wake Up interrupt, use high word bit 17.

enumerator kFLEXCAN_PNTimeoutWakeUpInterruptEnable

PN Timeout Wake Up interrupt, use high word bit 16. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

enumerator kFLEXCAN_ERxFifoUnderflowInterruptEnable

Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

enumerator kFLEXCAN_ERxFifoOverflowInterruptEnable

Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

enumerator kFLEXCAN_ERxFifoWatermarkInterruptEnable

Enhanced Rx FIFO Data Avilable interrupt, use high word bit 28.

enumerator kFLEXCAN_ERxFifoDataAvlInterruptEnable

enumerator kFLEXCAN_HostAccessNCErrorInterruptEnable

Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN_FlexCanAccessNCErrorInterruptEnable

FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN_HostOrFlexCanCErrorInterruptEnable

Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum _flexcan_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note: The CPU read action clears the bits corresponding to the FlEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using _flexcan_error_flags enumerations.

Values:

enumerator kFLEXCAN_ErrorOverrunFlag
Error Overrun Status.

enumerator kFLEXCAN_FDErrorIntFlag
CAN FD Error Interrupt Flag.

enumerator kFLEXCAN_BusoffDoneIntFlag
Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN_SynchFlag
CAN Synchronization Status.

enumerator kFLEXCAN_TxWarningIntFlag
Tx Warning Interrupt Flag.

enumerator kFLEXCAN_RxWarningIntFlag
Rx Warning Interrupt Flag.

enumerator kFLEXCAN_IdleFlag
FlexCAN In IDLE Status.

enumerator kFLEXCAN_FaultConfinementFlag
FlexCAN Fault Confinement State.

enumerator kFLEXCAN_TransmittingFlag
FlexCAN In Transmission Status.

enumerator kFLEXCAN_ReceivingFlag
FlexCAN In Reception Status.

enumerator kFLEXCAN_BusOffIntFlag
Bus Off Interrupt Flag.

enumerator kFLEXCAN_ErrorIntFlag
CAN Error Interrupt Flag.

enumerator kFLEXCAN_ErrorFlag

enumerator kFLEXCAN_PNMatchIntFlag
PN Matching Event Interrupt Flag.

enumerator kFLEXCAN_PNTimeoutIntFlag
PN Timeout Event Interrupt Flag.

enumerator kFLEXCAN_ERxFifoUnderflowIntFlag
Enhanced Rx FIFO underflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoOverflowIntFlag
Enhanced Rx FIFO overflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoWatermarkIntFlag
Enhanced Rx FIFO watermark Interrupt Flag.

enumerator kFLEXCAN_ERxFifoDataAvlIntFlag
Enhanced Rx FIFO data available Interrupt Flag.

enumerator kFLEXCAN_ERxFifoEmptyFlag
Enhanced Rx FIFO empty status.

enumerator kFLEXCAN_ERxFifoFullFlag
Enhanced Rx FIFO full status.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorIntFlag
Host Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag
FlexCAN Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_CorrectableErrorIntFlag
Correctable Error Interrupt Flag.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag
Host Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag
FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_CorrectableErrorOverrunFlag
Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_AllMemoryErrorIntFlag
All Memory Error Interrupt Flags.

enumerator kFLEXCAN_AllMemoryErrorFlag
All Memory Error Flags.

enum _flexcan_error_flags
FlexCAN error status flags.

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in _flexcan_flags enumerations to ditermine which error is generated.

Values:

enumerator kFLEXCAN_FDStuffingError
Stuffing Error.

enumerator kFLEXCAN_FDFormError
Form Error.

enumerator kFLEXCAN_FDCrcError
Cyclic Redundancy Check Error.

enumerator kFLEXCAN_FDBit0Error
Unable to send dominant bit.

enumerator kFLEXCAN_FDBit1Error
Unable to send recessive bit.

enumerator kFLEXCAN_TxErrorWarningFlag
Tx Error Warning Status.

enumerator kFLEXCAN_RxErrorWarningFlag
Rx Error Warning Status.

enumerator kFLEXCAN_StuffingError
Stuffing Error.

enumerator kFLEXCAN_FormError
Form Error.

enumerator kFLEXCAN_CrcError
Cyclic Redundancy Check Error.

enumerator kFLEXCAN_AckError
Received no ACK on transmission.

enumerator kFLEXCAN_Bit0Error
Unable to send dominant bit.

enumerator kFLEXCAN_Bit1Error
Unable to send recessive bit.

FlexCAN Legacy Rx FIFO status flags.

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Values:

enumerator kFLEXCAN_RxFifoOverflowFlag
Rx FIFO overflow flag.

enumerator kFLEXCAN_RxFifoWarningFlag
Rx FIFO almost full flag.

enumerator kFLEXCAN_RxFifoFrameAvlFlag
Frames available in Rx FIFO flag.

enum _flexcan_memory_error_type
FlexCAN Memory Error Type.

Values:

enumerator kFLEXCAN_CorrectableError
The memory error is correctable which means on bit error.

enumerator kFLEXCAN_NonCorrectableError
The memory error is non-correctable which means two bit errors.

enum _flexcan_memory_access_type
FlexCAN Memory Access Type.

Values:

enumerator kFLEXCAN_MoveOutFlexCanAccess
The memory error was detected during move-out FlexCAN access.

enumerator kFLEXCAN_MoveInAccess
The memory error was detected during move-in FlexCAN access.

enumerator kFLEXCAN_TxArbitrationAccess
The memory error was detected during Tx Arbitration FlexCAN access.

enumerator kFLEXCAN_RxMatchingAccess
The memory error was detected during Rx Matching FlexCAN access.

enumerator kFLEXCAN_MoveOutHostAccess
The memory error was detected during Rx Matching Host (CPU) access.

enum _flexcan_byte_error_syndrome
FlexCAN Memory Error Byte Syndrome.

Values:

enumerator kFLEXCAN_NoError
No bit error in this byte.

enumerator kFLEXCAN_ParityBits0Error
Parity bit 0 error in this byte.

enumerator kFLEXCAN_ParityBits1Error
Parity bit 1 error in this byte.

enumerator kFLEXCAN_ParityBits2Error
Parity bit 2 error in this byte.

enumerator kFLEXCAN_ParityBits3Error
Parity bit 3 error in this byte.

enumerator kFLEXCAN_ParityBits4Error
Parity bit 4 error in this byte.

enumerator kFLEXCAN_DataBits0Error
Data bit 0 error in this byte.

enumerator kFLEXCAN_DataBits1Error
Data bit 1 error in this byte.

enumerator kFLEXCAN_DataBits2Error
Data bit 2 error in this byte.

enumerator kFLEXCAN_DataBits3Error
Data bit 3 error in this byte.

enumerator kFLEXCAN_DataBits4Error
Data bit 4 error in this byte.

enumerator kFLEXCAN_DataBits5Error
Data bit 5 error in this byte.

enumerator kFLEXCAN_DataBits6Error
Data bit 6 error in this byte.

enumerator kFLEXCAN_DataBits7Error
Data bit 7 error in this byte.

enumerator kFLEXCAN_AllZeroError
All-zeros non-correctable error in this byte.

enumerator kFLEXCAN_AllOneError
All-ones non-correctable error in this byte.

enumerator kFLEXCAN_NonCorrectableErrors
Non-correctable error in this byte.

enum _flexcan_pn_match_source
FlexCAN Pretended Networking match source selection.

Values:

enumerator kFLEXCAN_PNMatSrcID
Message match with ID filtering.

enumerator kFLEXCAN_PNMatSrcIDAndData
Message match with ID filtering and payload filtering.

enum _flexcan_pn_match_mode

FlexCAN Pretended Networking mode match type.

Values:

enumerator kFLEXCAN_PNMatModeEqual

Match upon ID/Payload contents against an exact target value.

enumerator kFLEXCAN_PNMatModeGreater

Match upon an ID/Payload value greater than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeSmaller

Match upon an ID/Payload value smaller than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeRange

Match upon an ID/Payload value inside a range, greater than or equal to a specified lower limit, and smaller than or equal to a specified upper limit

typedef enum *_flexcan_frame_format* flexcan_frame_format_t

FlexCAN frame format.

typedef enum *_flexcan_frame_type* flexcan_frame_type_t

FlexCAN frame type.

typedef enum *_flexcan_clock_source* flexcan_clock_source_t

FlexCAN clock source.

Deprecated:

Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

typedef enum *_flexcan_wake_up_source* flexcan_wake_up_source_t

FlexCAN wake up source.

typedef enum *_flexcan_endianness* flexcan_endianness_t

FlexCAN payload endianness.

typedef enum *_flexcan_MB_timestamp_base* flexcan_MB_timestamp_base_t

FlexCAN timebase used for capturing 16-bit TIME_STAMP field of message buffer.

typedef enum *_flexcan_capture_point* flexcan_capture_point_t

FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

typedef enum *_flexcan_rx_fifo_filter_type* flexcan_rx_fifo_filter_type_t

FlexCAN Rx Fifo Filter type.

typedef enum *_flexcan_mb_size* flexcan_mb_size_t

FlexCAN Message Buffer Payload size.

typedef enum *_flexcan_efifo_dma_per_read_length* flexcan_efifo_dma_per_read_length_t

FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

typedef enum *_flexcan_rx_fifo_priority* flexcan_rx_fifo_priority_t

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

typedef enum *_flexcan_memory_error_type* flexcan_memory_error_type_t

FlexCAN Memory Error Type.

typedef enum *_flexcan_memory_access_type* flexcan_memory_access_type_t

FlexCAN Memory Access Type.

typedef enum *_flexcan_byte_error_syndrome* flexcan_byte_error_syndrome_t

FlexCAN Memory Error Byte Syndrome.

typedef struct *_flexcan_memory_error_report_status* flexcan_memory_error_report_status_t

FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN_GetMemoryErrorReportStatus() function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

typedef struct *_flexcan_frame* flexcan_frame_t

FlexCAN message frame structure.

typedef struct *_flexcan_fd_frame* flexcan_fd_frame_t

CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see _flexcan_fd_frame_length.

typedef struct *_flexcan_timing_config* flexcan_timing_config_t

FlexCAN protocol timing characteristic configuration structure.

typedef struct *_flexcan_config* flexcan_config_t

FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

typedef struct *_flexcan_rx_mb_config* flexcan_rx_mb_config_t

FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

typedef enum *_flexcan_pn_match_source* flexcan_pn_match_source_t

FlexCAN Pretended Networking match source selection.

typedef enum *_flexcan_pn_match_mode* flexcan_pn_match_mode_t

FlexCAN Pretended Networking mode match type.

typedef struct *_flexcan_pn_config* flexcan_pn_config_t

FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

typedef struct *_flexcan_rx_fifo_config* flexcan_rx_fifo_config_t

FlexCAN Legacy Rx FIFO configuration structure.

typedef struct *_flexcan_enhanced_rx_fifo_std_id_filter* flexcan_enhanced_rx_fifo_std_id_filter_t

FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

typedef struct *_flexcan_enhanced_rx_fifo_ext_id_filter* flexcan_enhanced_rx_fifo_ext_id_filter_t
FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

typedef struct *_flexcan_enhanced_rx_fifo_config* flexcan_enhanced_rx_fifo_config_t
FlexCAN Enhanced Rx FIFO configuration structure.

typedef struct *_flexcan_mb_transfer* flexcan_mb_transfer_t
FlexCAN Message Buffer transfer.

typedef struct *_flexcan_fifo_transfer* flexcan_fifo_transfer_t
FlexCAN Rx FIFO transfer.

typedef struct *_flexcan_handle* flexcan_handle_t
FlexCAN handle structure definition.

typedef void (*flexcan_transfer_callback_t)(CAN_Type *base, *flexcan_handle_t* *handle, *status_t* status, uint64_t result, void *userData)

FLEXCAN_WAIT_TIMEOUT

FLEXCAN_POLLING_TIMEOUT
Max loops to wait for polling transfer.

FLEXCAN_MODULE_TIMEOUT
Max loops to wait for FlexCAN register access complete.

DLC_LENGTH_DECODE(dlc)
FlexCAN frame length helper macro.

FLEXCAN_ID_STD(id)
FlexCAN Frame ID helper macro.

Standard Frame ID helper macro.

FLEXCAN_ID_EXT(id)
Extend Frame ID helper macro.

FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)
FlexCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)
Extend Rx Message Buffer Mask helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)
FlexCAN Legacy Rx FIFO Mask helper macro.

Standard Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)
Standard Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)
Standard Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id)
Standard Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id)
Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id)
Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)
Standard Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)
Extend Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)
Extend Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)
Extend Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)
Extend Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)
Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)
Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)
Extend Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)
FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)
Standard Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)
Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)
Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)
Standard Rx FIFO Filter helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)
Extend Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)
Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)
Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id)
Extend Rx FIFO Filter helper macro Type C lower part helper macro.

ENHANCED_RX_FIFO_FSCH(x)
FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.

RTR_STD_HIGH(x)

RTR_STD_LOW(x)

RTR_EXT(x)

ID_STD_LOW(id)

ID_STD_HIGH(id)

ID_EXT(id)

FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask)
Standard ID filter element with filter + mask scheme.

FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE(id_upper, rtr, id_lower, rtr_mask)
Standard ID filter element with filter range.

FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS(id1, rtr1, id2, rtr2)
Standard ID filter element with two filters without masks.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW(id, rtr)
Extended ID filter element with filter + mask scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH(id_mask, rtr_mask)
Extended ID filter element with filter + mask scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW(id_upper, rtr)
Extended ID filter element with range scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH(id_lower, rtr_mask)
Extended ID filter element with range scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW(id2, rtr2)
Extended ID filter element with two filters without masks low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH(id1, rtr1)
Extended ID filter element with two filters without masks high word.

FLEXCAN_PN_STD_MASK(id, rtr)
FlexCAN Pretended Networking ID Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_PN_EXT_MASK(id, rtr)
Extend Rx Message Buffer Mask helper macro.

FLEXCAN_PN_INT_MASK(x)
FlexCAN interrupt/status flag helper macro.

FLEXCAN_PN_INT_UNMASK(x)

FLEXCAN_PN_STATUS_MASK(x)

FLEXCAN_PN_STATUS_UNMASK(x)

FLEXCAN_EFIFO_INT_MASK(x)

FLEXCAN_EFIFO_INT_UNMASK(x)

FLEXCAN_EFIFO_STATUS_MASK(x)

FLEXCAN_EFIFO_STATUS_UNMASK(x)

FLEXCAN_MECR_INT_MASK(x)

FLEXCAN_MECR_INT_UNMASK(x)

FLEXCAN_MECR_STATUS_MASK(x)

FLEXCAN_MECR_STATUS_UNMASK(x)

FLEXCAN_ERROR_AND_STATUS_INT_FLAG

FLEXCAN_PNWAKE_UP_FLAG

FLEXCAN_WAKE_UP_FLAG

FLEXCAN_MEMORY_ERROR_INT_FLAG

FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG

FlexCAN Enhanced Rx FIFO base address helper macro.

E_RX_FIFO(base)

FLEXCAN_CALLBACK(x)

FlexCAN transfer callback function.

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_FLEXCAN_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

struct _flexcan_memory_error_report_status

#include <fsl_flexcan.h> FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of FLEXCAN_GetMemoryErrorReportStatus() function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

Public Members

flexcan_memory_error_type_t errorType

The type of memory error that giving rise to the report.

flexcan_memory_access_type_t accessType

The type of memory access that giving rise to the memory error.

uint16_t accessAddress

The address where memory error detected.

uint32_t errorData

The raw data word read from memory with error.

struct _flexcan_frame

#include <fsl_flexcan.h> FlexCAN message frame structure.

struct _flexcan_fd_frame

#include <fsl_flexcan.h> CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see _flexcan_fd_frame_length.

Public Members

uint32_t idhit

Note: ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using FLEXCAN_FixEnhancedRxFifoFrameIdHit API is recommended to ensure this idhit value is correct. CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).

uint32_t hrtimestamp

Note: HR timestamp offset is changed dynamically according to data length code (DLC). External 32-bit on-chip timer high-resolution timestamp.

struct _flexcan_timing_config

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

uint32_t preDivider

Classic CAN or CAN FD nominal phase bit rate prescaler.

uint32_t rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint32_t phaseSeg1

Classic CAN or CAN FD nominal phase Segment 1.

uint32_t phaseSeg2

Classic CAN or CAN FD nominal phase Segment 2.

uint32_t propSeg

Classic CAN or CAN FD nominal phase Propagation Segment.

uint32_t fpreDivider

CAN FD data phase bit rate prescaler.

uint32_t frJumpwidth

CAN FD data phase Re-sync Jump Width.

uint32_t fphaseSeg1

CAN FD data phase Phase Segment 1.

uint32_t fphaseSeg2

CAN FD data phase Phase Segment 2.

uint32_t fpropSeg
: CAN FD data phase Propagation Segment.

struct _flexcan_config
: *#include <fsl_flexcan.h>* FlexCAN module configuration structure.

Deprecated:
: Do not use the baudRate. It has been superceded bitRate

  Do not use the baudRateFD. It has been superceded bitRateFD

Public Members

flexcan_clock_source_t clkSrc
: Clock source for FlexCAN Protocol Engine.

flexcan_wake_up_source_t wakeupSrc
: Wake up source selection.

uint8_t maxMbNum
: The maximum number of Message Buffers used by user.

bool enableLoopBack
: Enable or Disable Loop Back Self Test Mode.

bool enableTimerSync
: Enable or Disable Timer Synchronization.

bool enableIndividMask
: Enable or Disable Rx Individual Mask and Queue feature.

bool disableSelfReception
: Enable or Disable Self Reflection.

bool enableListenOnlyMode
: Enable or Disable Listen Only Mode.

bool enableDoze
: Enable or Disable Doze Mode.

bool enablePretendedeNetworking
: Enable or Disable the Pretended Networking mode.

bool enableMemoryErrorControl
: Enable or Disable the memory errors detection and correction mechanism.

bool enableNonCorrectableErrorEnterFreeze
: Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.

bool enableTransceiverDelayMeasure
: Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.

bool enableRemoteRequestFrameStored
: true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response Frame.

flexcan_endianness_t payloadEndianness

Selects the byte order for the payload of transmit and receive frames, see flexcan_endianness_t.

bool enableExternalTimeTick

true: External time tick clocks the free-running timer. false: FlexCAN bit clock clocks the free-running timer.

flexcan_MB_timestamp_base_t captureTimeBase

Timebase of message buffer 16-bit TIME_STAMP field.

flexcan_capture_point_t capturePoint

Point in time when 32-bit timebase is captured during CAN frame.

struct __flexcan__rx__mb__config

#include <fsl_flexcan.h> FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of FLEXCAN_SetRxMbConfig() function. The FLEXCAN_SetRxMbConfig() function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

Public Members

uint32_t id

CAN Message Buffer Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

flexcan_frame_format_t format

CAN Frame Identifier format(Standard of Extend).

flexcan_frame_type_t type

CAN Frame Type(Data or Remote for classical CAN only).

struct __flexcan__pn__config

#include <fsl_flexcan.h> FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of FLEXCAN_SetPNConfig() function. The FLEXCAN_SetPNConfig() function is used to configure FlexCAN Networking work mode.

Public Members

bool enableTimeout

Enable or Disable timeout event trigger wakeup.

uint16_t timeoutValue

The timeout value that generates a wakeup event, the counter timer is incremented based on 64 times the CAN Bit Time unit.

bool enableMatch

Enable or Disable match event trigger wakeup.

flexcan_pn_match_source_t matchSrc

Selects the match source (ID and/or data match) to trigger wakeup.

uint8_t matchNum

The number of times a given message must match the predefined ID and/or data before generating a wakeup event, range in 0x1 ~ 0xFF.

flexcan_pn_match_mode_t idMatchMode
The ID match type.

flexcan_pn_match_mode_t dataMatchMode
The data match type.

uint32_t idLower
The ID target values 1 which used either for ID match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in ID match “range detection”.

uint32_t idUpper
The ID target values 2 which used only as the upper limit value in ID match “range detection” or used to store the ID mask in “equal to”.

uint8_t lengthLower
The lower limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

uint8_t lengthUpper
The upper limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

struct _flexcan_rx_fifo_config
#include <fsl_flexcan.h> FlexCAN Legacy Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable
Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

uint8_t idFilterNum
The FlexCAN Legacy Rx FIFO Filter elements quantity.

flexcan_rx_fifo_filter_type_t idFilterType
The FlexCAN Legacy Rx FIFO Filter type.

flexcan_rx_fifo_priority_t priority
The FlexCAN Legacy Rx FIFO receive priority.

struct _flexcan_enhanced_rx_fifo_std_id_filter
#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

Public Members

uint32_t filterType
FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1
CAN FD frame data length code (DLC), range see _flexcan_fd_frame_length, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use DLC_LENGTH_DECODE(length) macro to get the number of valid data bytes.

uint32_t std1
CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct __flexcan_enhanced_rx_fifo_ext_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see _flexcan_fd_frame_length, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use DLC_LENGTH_DECODE(length) macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct __flexcan_enhanced_rx_fifo_config

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to idFilterNum. There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupie 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

uint8_t idFilterPairNum

idFilterPairNum is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

uint8_t extendIdFilterNum

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, extendIdFilterNum need less than or equal to idFilterPairNum.

uint8_t fifoWatermark

(fifoWatermark + 1) is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

flexcan_efifo_dma_per_read_length_t dmaPerReadLength

Define the length of each read of the Enhanced RX FIFO element by the DAM, see _flexcan_fd_frame_length.

flexcan_rx_fifo_priority_t priority
The FlexCAN Enhanced Rx FIFO receive priority.

struct _flexcan_mb_transfer
#include <fsl_flexcan.h> FlexCAN Message Buffer transfer.

Public Members

flexcan_frame_t *frame
The buffer of CAN Message to be transfer.

uint8_t mbIdx
The index of Message buffer used to transfer Message.

struct _flexcan_fifo_transfer
#include <fsl_flexcan.h> FlexCAN Rx FIFO transfer.

Public Members

flexcan_fd_frame_t *framefd
The buffer of CAN Message to be received from Enhanced Rx FIFO.

flexcan_frame_t *frame
The buffer of CAN Message to be received from Legacy Rx FIFO.

size_t frameNum
Number of CAN Message need to be received from Legacy or Ehanced Rx FIFO.

struct _flexcan_handle
#include <fsl_flexcan.h> FlexCAN handle structure.

Public Members

flexcan_transfer_callback_t callback
Callback function.

void *userData
FlexCAN callback function parameter.

flexcan_frame_t *volatile mbFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN data from Message Buffers.

flexcan_fd_frame_t *volatile mbFDFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN FD data from Message Buffers.

flexcan_frame_t *volatile rxFifoFrameBuf
The buffer for received CAN data from Legacy Rx FIFO.

flexcan_fd_frame_t *volatile rxFifoFDFrameBuf
The buffer for received CAN FD data from Ehanced Rx FIFO.

size_t rxFifoFrameNum
The number of CAN messages remaining to be received from Legacy or Ehanced Rx FIFO.

size_t rxFifoTransferTotalNum
Total CAN Message number need to be received from Legacy or Ehanced Rx FIFO.

volatile uint8_t mbState[CAN_WORD1_COUNT]
Message Buffer transfer state.

volatile uint8_t rxFifoState
Rx FIFO transfer state.

volatile uint32_t timestamp[CAN_WORD1_COUNT]
Mailbox transfer timestamp.

struct byteStatus

Public Members

bool byteIsRead
The byte n (0~3) was read or not. The type of error and which bit in byte (n) is affected by the error.

struct ___unnamed18___

Public Members

uint32_t timestamp
FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length
CAN frame data length in bytes (Range: 0~8).

uint32_t type
CAN Frame Type(DATA or REMOTE).

uint32_t format
CAN Frame Identifier(STD or EXT format).

uint32_t ___pad0___
Reserved.

uint32_t idhit
CAN Rx FIFO filter hit id(This value is only used in Rx FIFO receive mode).

struct ___unnamed20___

Public Members

uint32_t id
CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

uint32_t ___pad0___
Reserved.

union ___unnamed22___

Public Members

struct __flexcan__frame

struct __flexcan__frame

struct ___unnamed24___

Public Members

uint32_t dataWord0

CAN Frame payload word0.

uint32_t dataWord1

CAN Frame payload word1.

struct ___unnamed26___

Public Members

uint8_t dataByte3

CAN Frame payload byte3.

uint8_t dataByte2

CAN Frame payload byte2.

uint8_t dataByte1

CAN Frame payload byte1.

uint8_t dataByte0

CAN Frame payload byte0.

uint8_t dataByte7

CAN Frame payload byte7.

uint8_t dataByte6

CAN Frame payload byte6.

uint8_t dataByte5

CAN Frame payload byte5.

uint8_t dataByte4

CAN Frame payload byte4.

struct ___unnamed28___

Public Members

uint32_t timestamp

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length

CAN FD frame data length code (DLC), range see _flexcan_fd_frame_length, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use DLC_LENGTH_DECODE(length) macro to get the number of valid data bytes.

uint32_t type

CAN Frame Type(DATA only).

uint32_t format

CAN Frame Identifier(STD or EXT format).

uint32_t srr

Substitute Remote request.

uint32_t esi

Error State Indicator.

uint32_t brs
: Bit Rate Switch.

uint32_t edl
: Extended Data Length.

struct ___unnamed30___

Public Members

uint32_t id
: CAN Frame Identifier, should be set using FLEXCAN_ID_EXT() or FLEXCAN_ID_STD() macro.

uint32_t ___pad0___
: Reserved.

union ___unnamed32___

Public Members

struct _flexcan_fd_frame

struct _flexcan_fd_frame

struct ___unnamed34___

Public Members

uint32_t dataWord[16]
: CAN FD Frame payload, 16 double word maximum.

struct ___unnamed36___

Public Members

uint8_t dataByte3
: CAN Frame payload byte3.

uint8_t dataByte2
: CAN Frame payload byte2.

uint8_t dataByte1
: CAN Frame payload byte1.

uint8_t dataByte0
: CAN Frame payload byte0.

uint8_t dataByte7
: CAN Frame payload byte7.

uint8_t dataByte6
: CAN Frame payload byte6.

uint8_t dataByte5
: CAN Frame payload byte5.

uint8_t dataByte4
CAN Frame payload byte4.

union ___unnamed38___

Public Members

struct _flexcan_config

struct _flexcan_config

struct ___unnamed40___

Public Members

uint32_t baudRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t baudRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

struct ___unnamed42___

Public Members

uint32_t bitRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t bitRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

union ___unnamed44___

Public Members

struct _flexcan_pn_config

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range

detection”.

struct _flexcan_pn_config

struct ___unnamed48___

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range

detection”.

Public Members

uint32_t lowerWord0
CAN Frame payload word0.

uint32_t lowerWord1
CAN Frame payload word1.

struct ___unnamed50___

Public Members

uint8_t lowerByte3
: CAN Frame payload byte3.

uint8_t lowerByte2
: CAN Frame payload byte2.

uint8_t lowerByte1
: CAN Frame payload byte1.

uint8_t lowerByte0
: CAN Frame payload byte0.

uint8_t lowerByte7
: CAN Frame payload byte7.

uint8_t lowerByte6
: CAN Frame payload byte6.

uint8_t lowerByte5
: CAN Frame payload byte5.

uint8_t lowerByte4
: CAN Frame payload byte4.

union ___unnamed46___

Public Members

struct __flexcan__pn__config

< The data target values 2 which used only as the upper limit value in data match “range

detection” or used to store the data mask in “equal to”.

struct __flexcan__pn__config

struct ___unnamed52___

< The data target values 2 which used only as the upper limit value in data match “range detection” or used to store the data mask in “equal to”.

Public Members

uint32_t upperWord0
: CAN Frame payload word0.

uint32_t upperWord1
: CAN Frame payload word1.

struct ___unnamed54___

Public Members

uint8_t upperByte3
: CAN Frame payload byte3.

uint8_t upperByte2
: CAN Frame payload byte2.

uint8_t upperByte1
: CAN Frame payload byte1.

uint8_t upperByte0
: CAN Frame payload byte0.

uint8_t upperByte7
: CAN Frame payload byte7.

uint8_t upperByte6
: CAN Frame payload byte6.

uint8_t upperByte5
: CAN Frame payload byte5.

uint8_t upperByte4
: CAN Frame payload byte4.

2.19 FlexCAN eDMA Driver

void FLEXCAN_TransferCreateHandleEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *rxFifoEdmaHandle)

Initializes the FlexCAN handle, which is used in transactional functions.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

void FLEXCAN_PrepareTransfConfiguration(CAN_Type *base, *flexcan_fifo_transfer_t* *pFifoXfer, *edma_transfer_config_t* *pEdmaConfig)

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

status_t FLEXCAN_StartTransferDatafromRxFIFO(CAN_Type *base, *flexcan_edma_handle_t* *handle, *edma_transfer_config_t* *pEdmaConfig)

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer confiugration and Start eDMA transfer

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferGetReceiveFifoCountEMDA(CAN_Type *base, *flexcan_edma_handle_t* *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXCAN_TransferAbortReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle)

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.

status_t FLEXCAN_TransferReceiveEnhancedFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives the CAN FD Message from the Enhanced Rx FIFO using eDMA.

This function receives the CAN FD Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA(CAN_Type *base, *flexcan_edma_handle_t* *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

FSL_FLEXCAN_EDMA_DRIVER_VERSION

FlexCAN EDMA driver version.

typedef struct *_flexcan_edma_handle* flexcan_edma_handle_t

typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, *flexcan_edma_handle_t* *handle, *status_t* status, void *userData)

FlexCAN transfer callback function.

struct _flexcan_edma_handle

#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.

Public Members

flexcan_edma_transfer_callback_t callback

Callback function.

void *userData

FlexCAN callback function parameter.

edma_handle_t *rxFifoEdmaHandle

The EDMA handler for Rx FIFO.

volatile uint8_t rxFifoState
: Rx FIFO transfer state.

size_t frameNum
: The number of messages that need to be received.

flexcan_fd_frame_t *framefd
: Point to the buffer of CAN Message to be received from Enhanced Rx FIFO.

2.20 FlexIO: FlexIO Driver

2.21 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)
: Gets the default configuration to configure the FlexIO module. The configuration can 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 FLEXO_Init to use the FlexIO module.

  Parameters

  - base – FlexIO peripheral base address

uint32_t FLEXIO_GetInstance(FLEXIO_Type *base)

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

void FLEXIO_Reset(FLEXIO_Type *base)

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

static inline void FLEXIO_Enable(FLEXIO_Type *base, bool enable)

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns

FlexIO pin input data

static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns

current State pointer

void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const *flexio_shifter_config_t* *shifterConfig)

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
.timerSelect = 0,
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address

- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const *flexio_timer_config_t* *timerConfig)

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index, *flexio_timer_decrement_source_t* clocksource)

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_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 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « 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 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « 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 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « 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 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « 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 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline 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 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « 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 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « 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 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « 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_ShifterTimerPolarityOnNegitive

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

2.22 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

2.23 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 FLEXO MCULCD master eDMA handle.

This function initializes the FLEXO MCULCD master eDMA handle which can be used for other FLEXO MCULCD transactional APIs. For a specified FLEXO 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

2.24 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

2.25 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

2.26 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)
Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
```

(continues on next page)

(continued from previous page)

```
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND _flexio_i2c_master_status_flags can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_I2C_RxFullFlag
  - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t* direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
  - kFLEXIO_I2C_Write: Transmit
  - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns
data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns
status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

status_t FLEXIO_I2C_MasterTransferGetCount(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.

- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_I2C_MasterTransferAbort(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak

NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout

Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable

Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable

Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag
Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag
Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag
Receive NAK flag.

enum _flexio_i2c_direction
Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write
Master send to slave.

enumerator kFLEXIO_I2C_Read
Master receive from slave.

typedef enum *_flexio_i2c_direction* flexio_i2c_direction_t
Direction of master transfer.

typedef struct *_flexio_i2c_type* FLEXIO_I2C_Type
Define FlexIO I2C master access structure typedef.

typedef struct *_flexio_i2c_master_config* flexio_i2c_master_config_t
Define FlexIO I2C master user configuration structure.

typedef struct *_flexio_i2c_master_transfer* flexio_i2c_master_transfer_t
Define FlexIO I2C master transfer structure.

typedef struct *_flexio_i2c_master_handle* flexio_i2c_master_handle_t
FlexIO I2C master handle typedef.

typedef void (*flexio_i2c_master_transfer_callback_t)(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *status_t* status, void *userData)
FlexIO I2C master transfer callback typedef.

I2C_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_i2c_type
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t SDAPinIndex
Pin select for I2C SDA.

uint8_t SCLPinIndex
Pin select for I2C SCL.

uint8_t shifterIndex[2]
Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]
Timer index used in FlexIO I2C.

uint32_t baudrate
Master transfer baudrate, used to calculate delay time.

struct __flexio_i2c_master_config
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster
Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze
Enable/disable FlexIO operation in doze mode.

bool enableInDebug
Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
Baud rate in Bps.

struct __flexio_i2c_master_transfer
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags
Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress
7-bit slave address.

flexio_i2c_direction_t direction
Transfer direction, read or write.

uint32_t subaddress
Sub address. Transferred MSB first.

uint8_t subaddressSize
Size of sub address.

uint8_t volatile *data
Transfer buffer.

volatile size_t dataSize
Transfer size.

struct __flexio_i2c_master_handle
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer
FlexIO I2C master transfer copy.

size_t transferSize
Total bytes to be transferred.

uint8_t state
Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback
Callback function called at transfer event. Callback function called at transfer event.

void *userData
Callback parameter passed to callback function.

bool needRestart
Whether master needs to send re-start signal.

2.27 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 FLEXO_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 Tx 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

2.28 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.

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 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_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 ENWRPinIndex

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.

2.29 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*2=6. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.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 SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge

First edge on SPSCK 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.

2.30 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 FLEXO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_UART_TxDataRegEmptyFlag
  - kFLEXIO_UART_RxEmptyFlag
  - kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UARt transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns
FlexIO UART transmit data register address.

static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns
FlexIO UART receive data register address.

static inline void FLEXIO_UART_EnableTxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_EnableRxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_Enable(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

static inline void FLEXIO_UART_WriteByte(*FLEXIO_UART_Type* *base, const uint8_t *buffer)

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

static inline void FLEXIO_UART_ReadByte(*FLEXIO_UART_Type* *base, uint8_t *buffer)

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the “background” receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn’t call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.

- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

void FLEXIO_UART_TransferStartRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void FLEXIO_UART_TransferStopRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferSendNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See flexio_uart_transfer_t.

Return values

- kStatus_Success – Successfully starts the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished, data not written to the TX register.

void FLEXIO_UART_TransferAbortSend(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetSendCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferReceiveNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to xfer->data. This function returns with the parameter receivedBytes set to 5. For the last 5 bytes, newly arrived data is saved from the

xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See flexio_uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

void FLEXIO_UART_TransferAbortReceive(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetReceiveCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy

Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy

Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle

UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR

ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun

UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout

UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport

Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char

FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar

7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar

8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar

9-bit data characters

enum _flexio_uart_interrupt_enable

Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable

Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
Define FlexIO UART status mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag
Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag
Receive buffer over run flag.

typedef enum *_flexio_uart_bit_count_per_char* flexio_uart_bit_count_per_char_t
FlexIO UART bit count per char.

typedef struct *_flexio_uart_type* FLEXIO_UART_Type
Define FlexIO UART access structure typedef.

typedef struct *_flexio_uart_config* flexio_uart_config_t
Define FlexIO UART user configuration structure.

typedef struct *_flexio_uart_transfer* flexio_uart_transfer_t
Define FlexIO UART transfer structure.

typedef struct *_flexio_uart_handle* flexio_uart_handle_t

typedef void (*flexio_uart_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *status_t* status, void *userData)
FlexIO UART transfer callback function.

UART_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_uart_type
#include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t TxPinIndex
Pin select for UART_Tx.

uint8_t RxPinIndex
Pin select for UART_Rx.

uint8_t shifterIndex[2]
Shifter index used in FlexIO UART.

uint8_t timerIndex[2]
Timer index used in FlexIO UART.

struct _flexio_uart_config
#include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

Public Members

bool enableUart
: Enable/disable FlexIO UART TX & RX.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode

bool enableInDebug
: Enable/disable FlexIO operation in debug mode

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

flexio_uart_bit_count_per_char_t bitCountPerChar
: number of bits, 7/8/9 -bit

struct __flexio_uart_transfer
: *#include <fsl_flexio_uart.h>* Define FlexIO UART transfer structure.

Public Members

size_t dataSize
: Transfer size

struct __flexio_uart_handle
: *#include <fsl_flexio_uart.h>* Define FLEXIO UART handle structure.

Public Members

const uint8_t *volatile txData
: Address of remaining data to send.

volatile size_t txDataSize
: Size of the remaining data to send.

uint8_t *volatile rxData
: Address of remaining data to receive.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t txDataSizeAll
: Total bytes to be sent.

size_t rxDataSizeAll
: Total bytes to be received.

uint8_t *rxRingBuffer
: Start address of the receiver ring buffer.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
Callback function.

void *userData
UART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

union ___unnamed88___

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.

2.31 INTM: Interrupt Monitor Driver

FSL_INTM_DRIVER_VERSION
INTM driver version.

enum _intm_monitor
Interrupt monitors.

Values:

enumerator kINTM_Monitor1

enumerator kINTM_Monitor2

enumerator kINTM_Monitor3

enumerator kINTM_Monitor4

typedef enum *_intm_monitor* intm_monitor_t
Interrupt monitors.

typedef struct *_intm_monitor_config* intm_monitor_config_t
INTM interrupt source configuration structure.

typedef struct *_intm_config* intm_config_t
INTM configuration structure.

void INTM_GetDefaultConfig(*intm_config_t* *config)
Fill in the INTM config struct with the default settings.

The default values are:

```
config[0].irqnumber = NotAvail_IRQn;
config[0].maxtimer = 1000U;
config[1].irqnumber = NotAvail_IRQn;
config[1].maxtimer = 1000U;
config[2].irqnumber = NotAvail_IRQn;
config[2].maxtimer = 1000U;
config[3].irqnumber = NotAvail_IRQn;
config[3].maxtimer = 1000U;
config->enable = false;
```

Parameters

- config – Pointer to user's INTM config structure.

void INTM_Init(INTM_Type *base, const *intm_config_t* *config)

Ungates the INTM clock and configures the peripheral for basic operation.

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

Parameters

- base – INTM peripheral base address
- config – Pointer to user's INTM config structure.

void INTM_Deinit(INTM_Type *base)

Disables the INTM module.

Parameters

- base – INTM peripheral base address

static inline void INTM_EnableCycleCount(INTM_Type *base, bool enable)

Enable the cycle count timer mode.

Monitor mode enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

Parameters

- base – INTM peripheral base address.
- enable – Enable the cycle count or not.

static inline void INTM_AckIrq(INTM_Type *base, IRQn_Type irq)

Interrupt Acknowledge.

Call this function in ISR to acknowledge interrupt.

Parameters

- base – INTM peripheral base address.
- irq – Handle interrupt number.

static inline void INTM_SetInterruptRequestNumber(INTM_Type *base, *intm_monitor_t* intms, IRQn_Type irq)

Interrupt Request Select.

This function is used to set the interrupt request number to monitor or check.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

- irq – Interrupt request number to monitor.

Returns
Select the interrupt request number to monitor.

static inline void INTM_SetMaxTime(INTM_Type *base, *intm_monitor_t* intms, uint32_t count)

Set the maximum count time.

This function is to set the maximum time from interrupt generation to confirmation.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.
- count – Timer maximum count.

static inline void INTM_ClearTimeCount(INTM_Type *base, *intm_monitor_t* intms)

Clear the timer period in units of count.

This function is used to clear the INTM_TIMERa register.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

static inline uint32_t INTM_GetTimeCount(INTM_Type *base, *intm_monitor_t* intms)

Gets the timer period in units of count.

This function is used to get the number of INTM clock cycles from interrupt request to confirmation interrupt processing. If this number exceeds the set maximum time, will be an error signal.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

static inline bool INTM_GetStatusFlags(INTM_Type *base, *intm_monitor_t* intms)

Interrupt monitor status.

This function indicates whether the INTM_TIMERa value has exceeded the INTM_LATENCYa value. If any interrupt source in INTM_TIMERa exceeds the programmed delay value, the monitor state can be cleared by calling the INTM_ClearTimeCount() API to clear the corresponding INTM_TIMERa register.

Parameters

- base – INTM peripheral base address.
- intms – Programmable interrupt monitors.

Returns
Whether INTM_TIMER value has exceeded INTM_LATENCY value. false:INTM_TIMER value has not exceeded the INTM_LATENCY value; true:INTM_TIMER value has exceeded the INTM_LATENCY value.

struct __intm_monitor_config

#include <fsl_intm.h> INTM interrupt source configuration structure.

Public Members

uint32_t maxtimer

Set the maximum timer

IRQn_Type irqnumber
Select the interrupt request number to monitor.

struct _intm_config
#include <fsl_intm.h> INTM configuration structure.

Public Members

bool enable
Interrupt source monitor config. enables the cycle count timer on a monitored interrupt request for comparison to the latency register.

2.32 Common Driver

FSL_COMMON_DRIVER_VERSION
common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE
No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART
Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART
Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART
Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART
Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART
Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO
Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI
Debug console based on QSCI.

MIN(a, b)
Computes the minimum of *a* and *b*.

MAX(a, b)
Computes the maximum of *a* and *b*.

UINT16_MAX
Max value of uint16_t type.

UINT32_MAX
Max value of uint32_t type.

SDK_ATOMIC_LOCAL_ADD(addr, val)
Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)
Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)
Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)
Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)
Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)
For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)
For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)
For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)
Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)
Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)
Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)
Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)
Macro to define a variable with alignbytes alignment

SDK_L1DCACHE_ALIGN(var)
Macro to define a variable with L1 d-cache line size alignment

SDK_SIZEALIGN(var, alignbytes)
Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value (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*.

MCUX_CS

AT_CACHE_LINE_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

CACHE_LINE_DATA

AT_QUICKACCESS_SECTION_CODE(func)

Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)

Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)

Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

MCUX_RAMFUNC

Function attribute to place function in RAM. For example, to place function my_func in ram, use like:

```
MCUX_RAMFUNC my_func
```

RAMFUNCTION_SECTION_CODE(func)

Place function in ram.

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI
Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART
Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C
Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C
Group number for LPI2C status codes.

enumerator kStatusGroup_UART
Group number for UART status codes.

enumerator kStatusGroup_I2C
Group number for UART status codes.

enumerator kStatusGroup_LPSCI
Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART
Group number for LPUART status codes.

enumerator kStatusGroup_SPI
Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPI
Group number for ECSPI status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator kStatusGroup_SDIOSLV
Group number for SDIOSLV 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 wil 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.

2.33 LCU: Logic Control Unit Driver

void LCU_Init(LCU_Type *base)

Initializes the LCU peripheral.

This function ungates the LCU clock.

Parameters

- base – LCU peripheral base address.

void LCU_Deinit(LCU_Type *base)

Deinitializes the LCU peripheral.

This function gates the LCU clock.

Parameters

- base – LCU peripheral base address.

void LCU_GetForceInputDefaultConfig(*lcu_force_config_t* *config)

Gets the default configuration for LCU force input.

This function initializes the LCU force input configuration structure to a default value. The default values are as follows. code config->CombinationEnable = kLCU_SoftwareOverride0; config->polarity = kLCU_ForceInputPolarityNoInverted; config->filter = false; endcode

Parameters

- config – Pointer to the configuration structure.

void LCU_GetOutputDefaultConfig(*lcu_output_config_t* *config)

Gets the default configuration for LCU output.

This function initializes the LCU output configuration structure to a default value. The default values are as follows. code config->outputEnable = false; config->softwareOverrideEnable = false; config->softwareOverridevalue = kLCU_SoftwareOverride0; config->softwareSyncSelect = kLCU_SyncInput0; config->SyncMode = kLCU_SoftwareImmediateSync; config->syncSelect = kLCU_SyncInput0; config->forceClearMode = kLCU_ClearWithDeassertion; config->outputPolarity = kLCU_OutputPolarityNotInverted; config->forceInputSentivity = 0U; config->riseFilter = 0U; config->fallFilter = 0U; config->lutValue = 0xFFFFU; endcode

Parameters

- config – Pointer to the configuration structure.

void LCU_ForceInit(LCU_Type *base, *lcu_force_inputs_t* forceInput, const *lcu_force_config_t* *forceInputConfig)

Initializes force input.

This function gates the LCU clock.

Parameters

- base – LCU peripheral base address.
- forceInput – LCU force input number.
- forceInputConfig – Pointer to the LCU force input configuration structure.

void LCU_OutputInit(LCU_Type *base, *lcu_outputs_t* output, const *lcu_output_config_t* *outputConfig)

Deinitializes the LCU output.

This function gates the LCU clock.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- outputConfig – Pointer to the LCU configuration structure.

static inline void LCU_SetLutValue(LCU_Type *base, *lcu_outputs_t* output, uint32_t lutValue)

Get LC logic inputs.

This function Sete lookup table with inputs for LC outputs. When inputs and lutvalue correspond to the table, lut generates lut event and output logic 1

static inline uint32_t LCU_GetInputs(LCU_Type *base)

Get LC logic inputs.

Parameters

- base – LCU peripheral base address.

Returns

The mask of LC inputs status. Users can refer to lcu_bitfields_map_t to confirm the meaning of mask

static inline uint32_t LCU_GetSoftwareOverrideInputs(LCU_Type *base)

Get software override inputs.

Parameters

- base – LCU peripheral base address.

Returns

The mask of software override inputs status. Users can refer to lcu_bitfields_map_t to confirm the meaning of mask

static inline uint32_t LCU_GetOutputs(LCU_Type *base)

Get LC logic outputs.

Parameters

- base – LCU peripheral base address.

Returns

The mask of LC outputs status. Users can refer to lcu_bitfields_map_t to confirm the meaning of mask

static inline uint32_t LCU_GetForceSensitivity(LCU_Type *base)

Get force sensitivity values.

Parameters

- base – LCU peripheral base address.

Returns

The mask of forced outputs status. Users can refer to lcu_bitfields_map_t to confirm the meaning of mask

static inline void LCU_EnableDebugMode(LCU_Type *base, uint32_t outputsMask, bool enable)

Enable lcu debug mode.

This function support LCU outputs can continue operation when the chip is in Debug mode.

Parameters

- base – LCU peripheral base address.
- outputsMask – Mask of debug mode for outputs associated with lcu_bitfields_map_t.
- enable – Switcher of LCU debug mode feature. “true” means to enable, “false” means not.

static inline void LCU_EnableSoftwareOverride(LCU_Type *base, uint32_t inputsMask, bool enable)

Enable software override of LC inputs.

Parameters

- base – LCU peripheral base address.
- inputsMask – Mask of inputs which enabl software override associated with lcu_bitfields_map_t.
- enable – Switcher of LCU software override feature. “true” means to enable, “false” means not.

static inline void LCU_SetSoftwareOverrideValue(LCU_Type *base, uint32_t inputMask, *lcu_software_override_t* value)

Set software override value of LC inputs.

This function Specifies the software override value for each LC input.

Parameters

- base – LCU peripheral base address.
- inputMask – Mask of inputs which set software override value associated with lcu_bitfields_map_t.
- value – software override value.

static inline void LCU_EnableOutput(LCU_Type *base, uint32_t outputsMask, bool enable)

Enable LC outputs.

Parameters

- base – LCU peripheral base address.
- outputsMask – Mask of outputs which enabled output associated with lcu_bitfields_map_t.
- enable – Switcher of LCU outputs feature. “true” means to enable, “false” means not.

static inline void LCU_SetOutputPolarity(LCU_Type *base, *lcu_outputs_t* output, *lcu_output_polarity_t* polarity)

Set the polarity of the outputs.

This function specifies the polarity of the outputs.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- polarity – The output polarity.

static inline void LCU_SetLutDma(LCU_Type *base, uint32_t lutMask, bool enable)

Set the LUT DMA requests generation.

This function enables the generation of a DMA request when an LUT event occurs

Parameters

- base – LCU peripheral base address.
- lutMask – Mask of LUT event associated with lcu_bitfields_map_t.
- enable – Switcher of DMA request by LUT event feature. “true” means to enable, “false” means not.

static inline void LCU_SetForceDma(LCU_Type *base, uint32_t forceMask, bool enable)

Set the force DMA requests generation.

This function enables the generation of a DMA request when an force event occurs

Parameters

- base – LCU peripheral base address.
- forceMask – Mask of force event associated with lcu_bitfields_map_t.
- enable – Switcher of DMA request by force event feature. “true” means to enable, “false” means not.

static inline void LCU_SetLutInterrupt(LCU_Type *base, uint32_t lutMask, bool enable)

Set the LUT interrupt requests generation.

This function enables the generation of a interrupt request when an LUT event occurs

Parameters

- base – LCU peripheral base address.
- lutMask – Mask of LUT event associated with lcu_bitfields_map_t.
- enable – Switcher of interrupt request by LUT event feature. “true” means to enable, “false” means not.

static inline void LCU_SetForceInterrupt(LCU_Type *base, uint32_t forceMask, bool enable)

Set the force interupt.

This function enables the generation of a interrupt request when an force event occurs

Parameters

- base – LCU peripheral base address.
- forceMask – Mask of force event associated with lcu_bitfields_map_t.
- enable – Switcher of interrupt request by force event feature. “true” means to enable, “false” means not.

static inline uint32_t LCU_GetLutInterruptStatus(LCU_Type *base)

Get LUT interrupt status.

Parameters

- base – LCU peripheral base address.

Returns

The mask of LUT interrupt status. Users can refer to lcu_bitfields_map_t to confirm the meaning of mask

static inline uint32_t LCU_GetForceInterruptStatus(LCU_Type *base)

Get force interrupt status.

Parameters

- base – LCU peripheral base address.

Returns

The mask of force interrupt status. Users can refer to lcu_bitfields_map_t to confirm the meaning of mask

static inline void LCU_ClearLutInterruptStatus(LCU_Type *base, uint32_t lutMask)

Clear the LUT interupt status.

This function Clear the LUT interupt status.

Parameters

- base – LCU peripheral base address.
- lutMask – Mask of LUT interrupt status associated with lcu_bitfields_map_t.

static inline void LCU_ClearForceInterruptStatus(LCU_Type *base, uint32_t forceMask)

Clears force interrupt status.

Parameters

- base – LCU peripheral base address.
- mask – Mask of force interrupt flags associated with lcu_bitfields_map_t.

static inline void LCU_MuxSelect(LCU_Type *base, *lcu_inputs_t* input, *lcu_muxcel_source_t* source)

Selects the source of the LC input.

Parameters

- base – LCU peripheral base address.
- input – LCU input number.
- source – The source source of the LC input.

static inline void LCU_SetRiseFilter(LCU_Type *base, *lcu_outputs_t* output, uint32_t value)

Set LCU rise filter.

This function specifies the number of consecutive clock cycles the filter output must be logic 1 before the output signal goes high.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- value – The value of rise filter.

static inline void LCU_SetFallFilter(LCU_Type *base, *lcu_outputs_t* output, uint32_t value)

Set LCU fall filter.

This function specifies the number of consecutive clock cycles the filter output must be logic 0 before the output signal goes low

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- value – The value of fall filter.

static inline void LCU_SoftwareSyncSelect(LCU_Type *base, *lcu_outputs_t* output, *lcu_sync_select_source_t* source)

Select software sync input.

This function selects which sync input to use for software synced mode.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- source – The sync source for the software synced mode of this LC.

static inline void LCU_SyncSelect(LCU_Type *base, *lcu_outputs_t* output, *lcu_sync_select_source_t* source)

Select sync input.

This function selects which sync input to use for synced mode.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- source – The sync source for the output of LCU.

static inline void LCU_SetForceClearMode(LCU_Type *base, *lcu_outputs_t* output, *lcu_force_clearing_mode_t* mode)

Set force clearing mode.

This function specifies the timing for clearing force events for output.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- mode – The mode of which timing to clear force events.

static inline void LCU_ForceInputSensitivity(LCU_Type *base, *lcu_outputs_t* output, uint32_t inputsMask)

Set force input Sensitivity.

Selects which force inputs affect specified output.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- inputsMask – The mask of input in one LC, reference to lcu_force_input_sensitivity_t.

static inline void LCU_SetSoftwareSyncMode(LCU_Type *base, *lcu_outputs_t* output, *lcu_software_sync_mode_t* mode)

Set force clearing mode.

This function specifies the software sync mode for the inputs.

Parameters

- base – LCU peripheral base address.
- output – LCU output number.
- mode – The mode of software sync for the output.

FSL_LCU_DRIVER_VERSION

LCU driver version 2.0.0.

enum _lcu_cell

_lcu_cell LCU logic cell flags.

Values:

enumerator kLCU_Lc0

Logic cell 0

enumerator kLCU_Lc1

Logic cell 1

enumerator kLCU_Lc2

Logic cell 2

enum _lcu_inputs

_lcu_inputs LCU inputs.

Values:

enumerator kLCU_Lc0Input0
LC0 input 0.

enumerator kLCU_Lc0Input1
LC0 input 1.

enumerator kLCU_Lc0Input2
LC0 input 2.

enumerator kLCU_Lc0Input3
LC0 input 3.

enumerator kLCU_Lc1Input0
LC1 input 0.

enumerator kLCU_Lc1Input1
LC1 input 1.

enumerator kLCU_Lc1Input2
LC1 input 2.

enumerator kLCU_Lc1Input3
LC1 input 3.

enumerator kLCU_Lc2Input0
LC2 input 0.

enumerator kLCU_Lc2Input1
LC2 input 1.

enumerator kLCU_Lc2Input2
LC2 input 2.

enumerator kLCU_Lc2Input3
LC2 input 3.

enum _lcu_outputs
_lcu_outputs LCU outputs.

Values:

enumerator kLCU_Lc0Output0
LC0 output 0.

enumerator kLCU_Lc0Output1
LC0 output 1.

enumerator kLCU_Lc0Output2
LC0 output 2.

enumerator kLCU_Lc0Output3
LC0 output 3.

enumerator kLCU_Lc1Output0
LC1 output 0.

enumerator kLCU_Lc1Output1
LC1 output 1.

enumerator kLCU_Lc1Output2
LC1 output 2.

enumerator kLCU_Lc1Output3
LC1 output 3.

enumerator kLCU_Lc2Output0
LC2 output 0.

enumerator kLCU_Lc2Output1
LC2 output 1.

enumerator kLCU_Lc2Output2
LC2 output 2.

enumerator kLCU_Lc2Output3
LC2 output 3.

enum _lcu_bitfields_map
_lcu_bitfields_map LCU inputs/outputs/states mask.

Values:

enumerator kLCU_Lc0IO0
LC0 input0/output0/state0 mask.

enumerator kLCU_Lc0IO1
LC0 input1/output1/state1 mask.

enumerator kLCU_Lc0IO2
LC0 input2/output2/state2 mask.

enumerator kLCU_Lc0IO3
LC0 input3/output3/state3 mask.

enumerator kLCU_Lc1IO0
LC1 input0/output0/state0 mask.

enumerator kLCU_Lc1IO1
LC1 input1/output1/state1 mask.

enumerator kLCU_Lc1IO2
LC1 input2/output2/state2 mask.

enumerator kLCU_Lc1IO3
LC1 input3/output3/state3 mask.

enumerator kLCU_Lc2IO0
LC2 input0/output0/state0 mask.

enumerator kLCU_Lc2IO1
LC2 input1/output1/state1 mask.

enumerator kLCU_Lc2IO2
LC2 input2/output2/state2 mask.

enumerator kLCU_Lc2IO3
LC2 input3/output3/state3 mask.

enum _lcu_lc_inputs
_lcu_lc_inputs LCU input for ecah LC.

Values:

enumerator kLCU_LcInput0
LCU input 0 for each LC.

enumerator kLCU_LcInput1
LCU input 1 for each LC.

enumerator kLCU_LcInput2
LCU input 2 for each LC.

enumerator kLCU_LcInput3
LCU input 3 for each LC.

enum _lcu_muxcel_source
_lcu_muxcel_source LCU MUX selected source.

Values:

enumerator kLCU_MuxSelLogic0
Select the logic0 as LC input.

enumerator kLCU_MuxSelInput0
Select the LCU input0 as LC input.

enumerator kLCU_MuxSelInput1
Select the LCU input1 as LC input.

enumerator kLCU_MuxSelInput2
Select the LCU input2 as LC input.

enumerator kLCU_MuxSelInput3
Select the LCU input3 as LC input.

enumerator kLCU_MuxSelInput4
Select the LCU input4 as LC input.

enumerator kLCU_MuxSelInput5
Select the LCU input5 as LC input.

enumerator kLCU_MuxSelInput6
Select the LCU input6 as LC input.

enumerator kLCU_MuxSelInput7
Select the LCU input7 as LC input.

enumerator kLCU_MuxSelInput8
Select the LCU input8 as LC input.

enumerator kLCU_MuxSelInput9
Select the LCU input9 as LC input.

enumerator kLCU_MuxSelInput10
Select the LCU input10 as LC input.

enumerator kLCU_MuxSelInput11
Select the LCU input11 as LC input.

enumerator kLCU_MuxSelOutput0
Select the LCU output0 as LC input.

enumerator kLCU_MuxSelOutput1
Select the LCU output1 as LC input.

enumerator kLCU_MuxSelOutput2
Select the LCU output2 as LC input.

enumerator kLCU_MuxSelOutput3
Select the LCU output3 as LC input.

enumerator kLCU_MuxSelOutput4
Select the LCU output4 as LC input.

enumerator kLCU_MuxSelOutput5
Select the LCU output5 as LC input.

enumerator kLCU_MuxSelOutput6
Select the LCU output6 as LC input.

enumerator kLCU_MuxSelOutput7
Select the LCU output7 as LC input.

enumerator kLCU_MuxSelOutput8
Select the LCU output8 as LC input.

enumerator kLCU_MuxSelOutput9
Select the LCU output9 as LC input.

enumerator kLCU_MuxSelOutput10
Select the LCU output10 as LC input.

enumerator kLCU_MuxSelOutput11
Select the LCU output11 as LC input.

enum _lcu_force_inputs
_lcu_force_inputs LCU force input.

Values:

enumerator kLCU_ForceInput0
LCU LC0 force inout0.

enumerator kLCU_ForceInput1
LCU LC0 force inout1.

enumerator kLCU_ForceInput2
LCU LC0 force inout2.

enumerator kLCU_ForceInput3
LCU LC0 force inout3.

enumerator kLCU_ForceInput4
LCU LC1 force inout0.

enumerator kLCU_ForceInput5
LCU LC1 force inout1.

enumerator kLCU_ForceInput6
LCU LC1 force inout2.

enumerator kLCU_ForceInput7
LCU LC1 force inout3.

enumerator kLCU_ForceInput8
LCU LC2 force inout0.

enumerator kLCU_ForceInput9
LCU LC2 force inout1.

enumerator kLCU_ForceInput10
LCU LC2 force inout2.

enumerator kLCU_ForceInput11
LCU LC2 force inout3.

enum __lcu_force_input_polarity
_lcu_force_input_polarity LCU force input polarity.

Values:

enumerator kLCU_ForceInputPolarityNoInverted
The polarity not inverted to the force input.

enumerator kLCU_ForceInputPolarityInverted
The polarity inverted to the force input.

enum __lcu_sync_select_source
_lcu_sync_select_source LCU sync mode selected source.

Values:

enumerator kLCU_SyncInput0
select LC sync input0 to use for output.

enumerator kLCU_SyncInput1
select LC sync input1 to use for output.

enum __lcu_software_sync_mode
_lcu_software_sync_mode LCU software sync mode.

Values:

enumerator kLCU_SoftwareImmediateSync
Software sync immediate for the inputs.

enumerator kLCU_SoftwareRiseEdgeSync
Software sync on rising edge of sync for the inputs.

enum __lcu_force_input_sensitivity
_lcu_force_Input_sensitivity LCU force input Sensitivity.

Values:

enumerator kLCU_ForceInput0Affect
Force input0 affect ouput.

enumerator kLCU_ForceInput1Affect
Force input1 affect ouput.

enumerator kLCU_ForceInput2Affect
Force input2 affect ouput.

enum __lcu_force_clearing_mode
_lcu_force_clearing_mode LCU force clearing mode.

Values:

enumerator kLCU_ClearWithDeassertion
Clear force events on deassertion of force inputs.

enumerator kLCU_ClearWithRisingSyncAfterDeassertion
Clear force events on rising sync after deassertion of force inputs.

enumerator kLCU_ClearWithStatusClearedAfterDeassertion
Clear force events when clear force event status after deassertion of force inputs.

enumerator kLCU_ClearWithRisingSyncAfterStatusClearedAndDeassertion
Clear force events on rising sync Deassertion en clear force event status after deassertion of force inputs..

enum __lcu_software_override
_lcu_software_override LCU software override value.

Values:

enumerator kLCU_SoftwareOverride0
LCU software override 0.

enumerator kLCU_SoftwareOverride1
LCU software override 1.

enum __lcu_output_polarity
_lcu_output_polarity LCU output polarity.

Values:

enumerator kLCU_OutputPolarityNotInverted
The polarity of the outputs not inverted.

enumerator kLCU_OutputPolarityInverted
The polarity of the outputs inverted.

typedef enum *_lcu_cell* lcu_cell_t
_lcu_cell LCU logic cell flags.

typedef enum *_lcu_inputs* lcu_inputs_t
_lcu_inputs LCU inputs.

typedef enum *_lcu_outputs* lcu_outputs_t
_lcu_outputs LCU outputs.

typedef enum *_lcu_bitfields_map* lcu_bitfields_map_t
_lcu_bitfields_map LCU inputs/outputs/states mask.

typedef enum *_lcu_lc_inputs* lcu_lc_inputs_t
_lcu_lc_inputs LCU input for ecah LC.

typedef enum *_lcu_muxcel_source* lcu_muxcel_source_t
_lcu_muxcel_source LCU MUX selected source.

typedef enum *_lcu_force_inputs* lcu_force_inputs_t
_lcu_force_inputs LCU force input.

typedef enum *_lcu_force_input_polarity* lcu_force_input_polarity_t
_lcu_force_input_polarity LCU force input polarity.

typedef enum *_lcu_sync_select_source* lcu_sync_select_source_t
_lcu_sync_select_source LCU sync mode selected source.

typedef enum *_lcu_software_sync_mode* lcu_software_sync_mode_t
_lcu_software_sync_mode LCU software sync mode.

typedef enum *_lcu_force_input_sensitivity* lcu_force_input_sensitivity_t
_lcu_force_Input_sensitivity LCU force input Sensitivity.

typedef enum *_lcu_force_clearing_mode* lcu_force_clearing_mode_t
_lcu_force_clearing_mode LCU force clearing mode.

typedef enum *_lcu_software_override* lcu_software_override_t
_lcu_software_override LCU software override value.

typedef enum *_lcu_output_polarity* lcu_output_polarity_t
_lcu_output_polarity LCU output polarity.

typedef struct *_lcu_force_config* lcu_force_config_t
LCU force configuration structure.

typedef struct *_lcu_output_config* lcu_output_config_t
LCU output configuration structure.

FSL_LCU_EACH_LC_IO_NUM
Macro used for calculate logic cell value.

FSL_LCU_GET_LC_VALUE(io)
Macro used for calculate logic cell value.

FSL_LCU_GET_LC_IO_VALUE(io)
Macro used for calculate which IO used in logic cell.

FSL_LCU_LC_OFFSET(cell)
Macro used for calculate mask offset in each logic cell.

FSL_LCU_FORCE_CONTROL_OFFSET(io)
Macro used for calculate force control offset in each IO.

struct _lcu_force_config
#include <fsl_lcu.h> LCU force configuration structure.

Public Members

bool CombinationEnable
Enable combinational force path.

lcu_force_input_polarity_t polarity
Force input polarity.

uint8_t filter
Force filter.

struct _lcu_output_config
#include <fsl_lcu.h> LCU output configuration structure.

Public Members

bool outputEnable
Enable LC output.

bool softwareOverrideEnable
Enable software override of input.

lcu_software_override_t softwareOverridevalue
Software override value for input.

lcu_sync_select_source_t softwareSyncSelect
Select sync input for software sync mode.

lcu_software_sync_mode_t SyncMode
Sync mode for input.

lcu_sync_select_source_t syncSelect
Select sync input for output.

lcu_force_clearing_mode_t forceClearMode
Timing for clearing force events for output.

lcu_output_polarity_t outputPolarity
Polarity of output.

uint32_t forceInputSentivity
Mask of force inputs that affect output @lcu_force_input_sensitivity_t

uint32_t riseFilter
Rise Filter.

uint32_t fallFilter
Fall Filter.

uint32_t lutValue
Value of LCU loolup table.

2.34 LPCMP: Low Power Analog Comparator Driver

void LPCMP_Init(LPCMP_Type *base, const *lpcmp_config_t* *config)
Initialize the LPCMP.

This function initializes the LPCMP module. The operations included are:

- Enabling the clock for LPCMP module.
- Configuring the comparator.
- Enabling the LPCMP module. Note: For some devices, multiple LPCMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the LPCMPs. Check the chip reference manual for the clock assignment of the LPCMP.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_config_t" structure.

void LPCMP_Deinit(LPCMP_Type *base)
De-initializes the LPCMP module.

This function de-initializes the LPCMP module. The operations included are:

- Disabling the LPCMP module.
- Disabling the clock for LPCMP module.

This function disables the clock for the LPCMP. Note: For some devices, multiple LPCMP instance shares the same clock gate. In this case, before disabling the clock for the LPCMP, ensure that all the LPCMP instances are not used.

Parameters

- base – LPCMP peripheral base address.

void LPCMP_GetDefaultConfig(*lpcmp_config_t* *config)
Gets an available pre-defined settings for the comparator's configuration.

This function initializes the comparator configuration structure to these default values:

```
config->enableStopMode      = false;
config->enableOutputPin     = false;
config->enableCmpToDacLink  = false;
config->useUnfilteredOutput = false;
config->enableInvertOutput  = false;
config->hysteresisMode      = kLPCMP_HysteresisLevel0;
config->powerMode           = kLPCMP_LowSpeedPowerMode;
config->functionalSourceClock = kLPCMP_FunctionalClockSource0;
config->plusInputSrc        = kLPCMP_PlusInputSrcMux;
config->minusInputSrc       = kLPCMP_MinusInputSrcMux;
```

Parameters

- config – Pointer to "lpcmp_config_t" structure.

static inline void LPCMP_Enable(LPCMP_Type *base, bool enable)

Enable/Disable LPCMP module.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable the module, and "false" means disable the module.

void LPCMP_SetInputChannels(LPCMP_Type *base, uint32_t positiveChannel, uint32_t negativeChannel)

Select the input channels for LPCMP. This function determines which input is selected for the negative and positive mux.

Parameters

- base – LPCMP peripheral base address.
- positiveChannel – Positive side input channel number. Available range is 0-7.
- negativeChannel – Negative side input channel number. Available range is 0-7.

static inline void LPCMP_EnableDMA(LPCMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events. Normally, the LPCMP generates a CPU interrupt if there is a rising/falling event. When DMA support is enabled and the rising/falling interrupt is enabled , the rising/falling event forces a DMA transfer request rather than a CPU interrupt instead.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable DMA support, and "false" means disable DMA support.

void LPCMP_SetFilterConfig(LPCMP_Type *base, const *lpcmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_filter_config_t" structure.

void LPCMP_SetDACConfig(LPCMP_Type *base, const *lpcmp_dac_config_t* *config)

Configure the internal DAC module.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_dac_config_t" structure. If config is "NULL", disable internal DAC.

static inline void LPCMP_EnableInterrupts(LPCMP_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See "_lpcmp_interrupt_enable".

static inline void LPCMP_DisableInterrupts(LPCMP_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See "_lpcmp_interrupt_enable".

static inline uint32_t LPCMP_GetStatusFlags(LPCMP_Type *base)

Get the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.

Returns

Mask value for the asserted flags. See "_lpcmp_status_flags".

static inline void LPCMP_ClearStatusFlags(LPCMP_Type *base, uint32_t mask)

Clear the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for the flags. See "_lpcmp_status_flags".

static inline void LPCMP_EnableWindowMode(LPCMP_Type *base, bool enable)

Enable/Disable window mode.When any windowed mode is active, COUTA is clocked by the bus clock whenever WINDOW = 1. The last latched value is held when WINDOW = 0. The optionally inverted comparator output COUT_RAW is sampled on every bus clock when WINDOW=1 to generate COUTA.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable window mode, and "false" means disable window mode.

void LPCMP_SetWindowControl(LPCMP_Type *base, const *lpcmp_window_control_config_t* *config)

Configure the window control, users can use this API to implement operations on the window, such as inverting the window signal, setting the window closing event(only valid in windowing mode), and setting the COUTA signal after the window is closed(only valid in windowing mode).

Parameters

- base – LPCMP peripheral base address.
- config – Pointer "lpcmp_window_control_config_t" structure.

void LPCMP_SetRoundRobinConfig(LPCMP_Type *base, const *lpcmp_roundrobin_config_t* *config)

Configure the roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer "lpcmp_roundrobin_config_t" structure.

static inline void LPCMP_EnableRoundRobinMode(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable roundrobin mode, and "false" means disable roundrobin mode.

void LPCMP_SetRoundRobinInternalTimer(LPCMP_Type *base, uint32_t value)

brief Configure the roundrobin internal timer reload value.

param base LPCMP peripheral base address. param value RoundRobin internal timer reload value, allowed range:0x0UL-0xFFFFFFFUL.

static inline void LPCMP_EnableRoundRobinInternalTimer(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin internal timer, note that this function is only valid when using the internal trigger source.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable roundrobin internal timer, and "false" means disable roundrobin internal timer.

static inline void LPCMP_SetPreSetValue(LPCMP_Type *base, uint8_t mask)

Set preset value for all channels, users can set all channels' preset vaule through this API, for example, if the mask set to 0x03U means channel0 and channel2's preset value set to 1U and other channels' preset value set to 0U.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetComparisonResult(LPCMP_Type *base)

Get comparison results for all channels, users can get all channels' comparison results through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' comparison result.

static inline void LPCMP_ClearInputChangedFlags(LPCMP_Type *base, uint8_t mask)

Clear input changed flags for single channel or multiple channels, users can clear input changed flag of a single channel or multiple channels through this API, for example, if the mask set to 0x03U means clear channel0 and channel2's input changed flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetInputChangedFlags(LPCMP_Type *base)

Get input changed flags for all channels, Users can get all channels' input changed flags through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' changed flag.

FSL_LPCMP_DRIVER_VERSION

LPCMP driver version 2.3.2.

enum _lpcmp_status_flags

LPCMP status falgs mask.

Values:

enumerator kLPCMP_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kLPCMP_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kLPCMP_OutputRoundRobinEventFlag

Detects when any channel's last comparison result is different from the pre-set value in trigger mode.

enumerator kLPCMP_OutputAssertEventFlag

Return the current value of the analog comparator output. The flag does not support W1C.

enum _lpcmp_interrupt_enable

LPCMP interrupt enable/disable mask.

Values:

enumerator kLPCMP_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kLPCMP_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enumerator kLPCMP_RoundRobinInterruptEnable

Comparator round robin mode interrupt occurred when the comparison result changes for a given channel.

enum _lpcmp_hysteresis_mode

LPCMP hysteresis mode. See chip data sheet to get the actual hystersis value with each level.

Values:

enumerator kLPCMP_HysteresisLevel0

The hard block output has level 0 hysteresis internally.

enumerator kLPCMP_HysteresisLevel1

The hard block output has level 1 hysteresis internally.

enumerator kLPCMP_HysteresisLevel2

The hard block output has level 2 hysteresis internally.

enumerator kLPCMP_HysteresisLevel3
The hard block output has level 3 hysteresis internally.

enum _lpcmp_power_mode
LPCMP nano mode.

Values:

enumerator kLPCMP_LowSpeedPowerMode
Low speed comparison mode is selected.

enumerator kLPCMP_HighSpeedPowerMode
High speed comparison mode is selected.

enumerator kLPCMP_NanoPowerMode
Nano power comparator is enabled.

enum _lpcmp_dac_reference_voltage_source
Internal DAC reference voltage source.

Values:

enumerator kLPCMP_VrefSourceVin1
vrefh_int is selected as resistor ladder network supply reference Vin.

enumerator kLPCMP_VrefSourceVin2
vrefh_ext is selected as resistor ladder network supply reference Vin.

enum _lpcmp_functional_source_clock
LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

Values:

enumerator kLPCMP_FunctionalClockSource0
Select functional mode clock source0.

enumerator kLPCMP_FunctionalClockSource1
Select functional mode clock source1.

enumerator kLPCMP_FunctionalClockSource2
Select functional mode clock source2.

enumerator kLPCMP_FunctionalClockSource3
Select functional mode clock source3.

enum _lpcmp_couta_signal
Set the COUTA signal value when the window is closed.

Values:

enumerator kLPCMP_COUTASignalNoSet
NO set the COUTA signal value when the window is closed.

enumerator kLPCMP_COUTASignalLow
Set COUTA signal low(0) when the window is closed.

enumerator kLPCMP_COUTASignalHigh
Set COUTA signal high(1) when the window is closed.

enum _lpcmp_close_window_event

Set COUT event, which can close the active window in window mode.

Values:

enumerator kLPCMP_CLoseWindowEventNoSet

No Set COUT event, which can close the active window in window mode.

enumerator kLPCMP_CloseWindowEventRisingEdge

Set rising edge COUT signal as COUT event.

enumerator kLPCMP_CloseWindowEventFallingEdge

Set falling edge COUT signal as COUT event.

enumerator kLPCMP_CLoseWindowEventBothEdge

Set both rising and falling edge COUT signal as COUT event.

enum _lpcmp_roundrobin_fixedmuxport

LPCMP round robin mode fixed mux port.

Values:

enumerator kLPCMP_FixedPlusMuxPort

Fixed plus mux port.

enumerator kLPCMP_FixedMinusMuxPort

Fixed minus mux port.

enum _lpcmp_roundrobin_clock_source

LPCMP round robin mode clock source selection.

Note: In different devices,the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

Values:

enumerator kLPCMP_RoundRobinClockSource0

Select roundrobin mode clock source0.

enumerator kLPCMP_RoundRobinClockSource1

Select roundrobin mode clock source1.

enumerator kLPCMP_RoundRobinClockSource2

Select roundrobin mode clock source2.

enumerator kLPCMP_RoundRobinClockSource3

Select roundrobin mode clock source3.

enum _lpcmp_roundrobin_trigger_source

LPCMP round robin mode trigger source.

Values:

enumerator kLPCMP_TriggerSourceExternally

Select external trigger source.

enumerator kLPCMP_TriggerSourceInternally

Select internal trigger source.

enum _lpcmp_plus_input_src

LPCMP plus input source.

Values:

enumerator kLPCMP_PlusInputSrcDac
LPCMP plus input source from the internal 8-bit DAC output.

enumerator kLPCMP_PlusInputSrcMux
LPCMP plus input source from the analog 8-1 mux.

enum _lpcmp_minus_input_src
LPCMP minus input source.

Values:

enumerator kLPCMP_MinusInputSrcDac
LPCMP minus input source from the internal 8-bit DAC output.

enumerator kLPCMP_MinusInputSrcMux
LPCMP minus input source from the analog 8-1 mux.

typedef enum *_lpcmp_hysteresis_mode* lpcmp_hysteresis_mode_t
LPCMP hysteresis mode. See chip data sheet to get the actual hystersis value with each level.

typedef enum *_lpcmp_power_mode* lpcmp_power_mode_t
LPCMP nano mode.

typedef enum *_lpcmp_dac_reference_voltage_source* lpcmp_dac_reference_voltage_source_t
Internal DAC reference voltage source.

typedef enum *_lpcmp_functional_source_clock* lpcmp_functional_source_clock_t
LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

typedef enum *_lpcmp_couta_signal* lpcmp_couta_signal_t
Set the COUTA signal value when the window is closed.

typedef enum *_lpcmp_close_window_event* lpcmp_close_window_event_t
Set COUT event, which can close the active window in window mode.

typedef enum *_lpcmp_roundrobin_fixedmuxport* lpcmp_roundrobin_fixedmuxport_t
LPCMP round robin mode fixed mux port.

typedef enum *_lpcmp_roundrobin_clock_source* lpcmp_roundrobin_clock_source_t
LPCMP round robin mode clock source selection.

Note: In different devices,the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

typedef enum *_lpcmp_roundrobin_trigger_source* lpcmp_roundrobin_trigger_source_t
LPCMP round robin mode trigger source.

typedef struct *_lpcmp_filter_config* lpcmp_filter_config_t
Configure the filter.

typedef enum *_lpcmp_plus_input_src* lpcmp_plus_input_src_t
LPCMP plus input source.

typedef enum *_lpcmp_minus_input_src* lpcmp_minus_input_src_t
LPCMP minus input source.

typedef struct *_lpcmp_dac_config* lpcmp_dac_config_t
configure the internal DAC.

typedef struct *_lpcmp_config* lpcmp_config_t
Configures the comparator.

typedef struct *_lpcmp_window_control_config* lpcmp_window_control_config_t
Configure the window mode control.

typedef struct *_lpcmp_roundrobin_config* lpcmp_roundrobin_config_t
Configure the round robin mode.

LPCMP_CCR1_COUTA_CFG_MASK

LPCMP_CCR1_COUTA_CFG_SHIFT

LPCMP_CCR1_COUTA_CFG(x)

LPCMP_CCR1_EVT_SEL_CFG_MASK

LPCMP_CCR1_EVT_SEL_CFG_SHIFT

LPCMP_CCR1_EVT_SEL_CFG(x)

struct _lpcmp_filter_config
#include <fsl_lpcmp.h> Configure the filter.

Public Members

bool enableSample
Decide whether to use the external SAMPLE as a sampling clock input.

uint8_t filterSampleCount
Filter Sample Count. Available range is 1-7; 0 disables the filter.

uint8_t filterSamplePeriod
Filter Sample Period. The divider to the bus clock. Available range is 0-255. The sampling clock must be at least 4 times slower than the system clock to the comparator. So if enableSample is “false”, filterSamplePeriod should be set greater than 4.

struct _lpcmp_dac_config
#include <fsl_lpcmp.h> configure the internal DAC.

Public Members

bool enableLowPowerMode
Decide whether to enable DAC low power mode.

lpcmp_dac_reference_voltage_source_t referenceVoltageSource
Internal DAC supply voltage reference source.

uint8_t DACValue
Value for the DAC Output Voltage. Different devices has different available range, for specific values, please refer to the reference manual.

struct _lpcmp_config
#include <fsl_lpcmp.h> Configures the comparator.

Public Members

bool enableStopMode
: Decide whether to enable the comparator when in STOP modes.

bool enableCmpToDacLink
: Controls the link from the CMP enable to the DAC enable.

bool enableOutputPin
: Decide whether to enable the comparator is available in selected pin.

bool useUnfilteredOutput
: Decide whether to use unfiltered output.

bool enableInvertOutput
: Decide whether to inverts the comparator output.

lpcmp_hysteresis_mode_t hysteresisMode
: LPCMP hysteresis mode.

lpcmp_power_mode_t powerMode
: LPCMP power mode.

lpcmp_functional_source_clock_t functionalSourceClock
: Select LPCMP functional mode clock source.

lpcmp_plus_input_src_t plusInputSrc
: Select LPCMP plus input source.

lpcmp_minus_input_src_t minusInputSrc
: Select LPCMP minus input source.

struct __lpcmp__window__control__config
: *#include <fsl_lpcmp.h>* Configure the window mode control.

Public Members

bool enableInvertWindowSignal
: True: enable invert window signal, False: disable invert window signal.

lpcmp_couta_signal_t COUTASignal
: Decide whether to define the COUTA signal value when the window is closed.

lpcmp_close_window_event_t closeWindowEvent
: Decide whether to select COUT event signal edge defines a COUT event to close window.

struct __lpcmp__roundrobin__config
: *#include <fsl_lpcmp.h>* Configure the round robin mode.

Public Members

uint8_t initDelayModules
: Comparator and DAC initialization delay modulus, See Reference Manual and DataSheet for specific value.

uint8_t sampleClockNumbers
: Specify the number of the round robin clock cycles(0~3) to wait after scanning the active channel before sampling the channel's comparison result.

uint8_t channelSampleNumbers

Specify the number of samples for one channel, note that channelSampleNumbers must not smaller than sampleTimeThreshhold.

uint8_t sampleTimeThreshhold

Specify that for one channel, when (sampleTimeThreshhold + 1) sample results are “1”,the final result is “1”, otherwise the final result is “0”, note that the sampleTimeThreshhold must not be larger than channelSampleNumbers.

lpcmp_roundrobin_clock_source_t roundrobinClockSource

Decide which clock source to choose in round robin mode.

lpcmp_roundrobin_trigger_source_t roundrobinTriggerSource

Decide which trigger source to choose in round robin mode.

lpcmp_roundrobin_fixedmuxport_t fixedMuxPort

Decide which mux port to choose as fixed channel in round robin mode.

uint8_t fixedChannel

Indicate which channel of the fixed mux port is used in round robin mode.

uint8_t checkerChannelMask

Indicate which channel of the non-fixed mux port to check its voltage value in round robin mode, for example, if checkerChannelMask set to 0x11U means select channel 0 and channel 4 as checker channel.

2.35 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.

2.36 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;
```

(continues on next page)

(continued from previous page)

```
masterConfig->hostRequest.enable      = false;
masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to lpi2c_master_config_t.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

Convert provided flags to status code, and clear any errors if present.

Parameters

- base – The LPI2C peripheral base address.
- status – Current status flags value that will be checked.

Return values

- kStatus_Success –
- kStatus_LPI2C_PinLowTimeout –
- kStatus_LPI2C_ArbitrationLost –

- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag

- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags().

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_LPI2C_Nak.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag
Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag
Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag
End Packet flag

enumerator kLPI2C_MasterStopDetectFlag
Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag
NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag
Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag
FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag
Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag
Data match flag

enumerator kLPI2C_MasterBusyFlag
Master busy flag

enumerator kLPI2C_MasterBusBusyFlag
Bus busy flag

enumerator kLPI2C_MasterClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
Errors to check for.

enum _lpi2c_direction
Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write
Master transmit.

enumerator kLPI2C_Read
Master receive.

enum _lpi2c_master_pin_config
LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain
LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Values:

enumerator kLPI2C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag
Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t
Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t
LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t
LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t
LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t

LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t

LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

LPI2C master descriptor of the transfer.

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

LPI2C master handle of the transfer.

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base
The LPI2C peripheral base address.

Param handle
Pointer to the LPI2C master driver handle.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct __lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable
: Enable transfers to continue when halted in debug mode.

bool ignoreAck
: Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig
: The pin configuration option.

uint32_t baudRate_Hz
: Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns
: Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns
: Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest
: Host request options.

struct __lpi2c_match_config
: *#include <fsl_lpi2c.h>* LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
: Data match configuration setting.

bool rxDataMatchOnly
: When set to true, received data is ignored until a successful match.

uint32_t match0
: Match value 0.

uint32_t match1
: Match value 1.

struct __lpi2c_master_transfer
: *#include <fsl_lpi2c.h>* Non-blocking transfer descriptor structure.

  This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
: Bit mask of options for the transfer. See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefaultFlag for normal transfers.

uint16_t slaveAddress
: The 7-bit slave address.

lpi2c_direction_t direction
: Either kLPI2C_Read or kLPI2C_Write.

uint32_t subaddress
: Sub address. Transferred MSB first.

size_t subaddressSize
: Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
: Pointer to data to transfer.

size_t dataSize
: Number of bytes to transfer.

struct __lpi2c_master_handle
: *#include <fsl_lpi2c.h>* Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
: Transfer state machine current state.

uint16_t remainingBytes
: Remaining byte count in current state.

uint8_t *buf
: Buffer pointer for current state.

uint16_t commandBuffer[6]
: LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer
: Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback
: Callback function pointer.

void *userData
: Application data passed to callback.

struct hostRequest

Public Members

bool enable
: Enable host request.

lpi2c_host_request_source_t source
: Host request source.

lpi2c_host_request_polarity_t polarity
: Host request pin polarity.

2.37 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.

2.38 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave              = true;
slaveConfig->address0                 = 0U;
slaveConfig->address1                 = 0U;
slaveConfig->addressMatchMode         = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable         = true;
slaveConfig->filterEnable             = true;
slaveConfig->enableGeneralCall        = false;
slaveConfig->sclStall.enableAck       = false;
slaveConfig->sclStall.enableTx        = true;
slaveConfig->sclStall.enableRx        = true;
slaveConfig->sclStall.enableAddress   = true;
slaveConfig->ignoreAck                = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns  = 0;
slaveConfig->sclGlitchFilterWidth_ns  = 0;
slaveConfig->dataValidDelay_ns        = 0;
slaveConfig->clockHoldTime_ns         = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t.

void LPI2C_SlaveInit(LPI2C_Type *base, const *lpi2c_slave_config_t* *slaveConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use LPI2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_slave_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_slave_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags().

static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the kLPI2C_SlaveTransmitAckFlag is asserted. This only happens if you enable the sclStall.enableAck field of the lpi2c_slave_config_t configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.
- ackOrNack – Pass true for an ACK or false for a NAK.

static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL,false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns
Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns
Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of lpi2c_slave_transfer_event_t enumerators for the events you wish to receive. The kLPI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kLPI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together lpi2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kLPI2C_SlaveAllEvents to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_LPI2C_Busy – Slave transfers have already been started on this handle.

status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

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.

2.39 LPSPI: Low Power Serial Peripheral Interface

2.40 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_SetPCSContinous(LPSPI_Type *base, bool IsContinous)

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.
- IsContinous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame

size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a “0” if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, *lpspi_delay_type_t* whichDelay)

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.

- scaler – The 8-bit delay value 0x00 to 0xFF (255).
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.

uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, *lpspi_delay_type_t* whichDelay, uint32_t srcClock_Hz)

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler).

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nano-seconds.
- whichDelay – The desired delay to configuration, which must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.
- data – The data word to be sent.

static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The data read from the data buffer.

void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_master_transfer_callback_t* callback, void *userData)

Initializes the LPSPI master handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_master_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns
true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag

Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag

Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag

Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag

Data Match flag

enumerator kLPSPI_ModuleBusyFlag

Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable
LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable
Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0
PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity
LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity
LPSPI clock polarity configuration.

Values:

enumerator 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 kLPSI_MatchDisabled

LPSPI Match Disabled.

enumerator kLPSI_1stWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordEqualsM0and2ndWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_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_MasterByteSwapyou 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.

2.41 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 uesed.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.
- kStatus_InvalidArgument – The transfer structure is invalid.

Returns
Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, size_t *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_slave_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle)

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.

- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.

status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, size_t *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns

status of status_t.

struct __lpspi_master_edma_handle

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous

Is PCS continuous in transfer.

volatile bool isByteSwap

A flag that whether should byte swap.

volatile uint8_t fifoSize

FIFO dataSize.

volatile uint8_t rxWatermark

Rx watermark.

volatile uint8_t bytesEachWrite

Bytes for each write TDR.

volatile uint8_t bytesEachRead

Bytes for each read RDR.

volatile uint8_t bytesLastRead

Bytes for last read RDR.

volatile bool isThereExtraRxBytes

Is there extra RX byte.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t txRemainingByteCount

Number of bytes remaining to send.

volatile size_t rxRemainingByteCount

Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

edma_tcd_t *lastTimeTCD
: Pointer to the lastTime TCD

bool isMultiDMATransmit
: Is there multi DMA transmit

volatile uint8_t dmaTransmitTime
: DMA Transfer times.

uint32_t lastTimeDataBytes
: DMA transmit last Time data Bytes

uint32_t dataBytesEveryTime
: Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

edma_transfer_config_t transferConfigRx
: Config of DMA rx channel.

edma_transfer_config_t transferConfigTx
: Config of DMA tx channel.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

uint32_t transmitCommand
: Used to write TCR for DMA purpose.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
: SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
: *#include <fsl_lpspi_edma.h>* LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

volatile uint8_t bytesLastRead
: Bytes for last read RDR.

volatile bool isThereExtraRxBytes
: Is there extra RX byte.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

volatile uint8_t state
: LPSPI transfer state.

uint32_t errorCount
: Error count for slave transfer.

lpspi_slave_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg

edma_tcd_t lpspiSoftwareTCD[2]
: SoftwareTCD, internal used

2.42 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.43 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, flase 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 receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in lpuart_handle_t.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators _lpuart_flags. To check for a specific status, compare the return value with enumerators in the _lpuart_flags. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of _lpuart_interrupt_enable.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the LPUART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the kStatus_LPUART_TxIdle as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferStartRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART_TransferStopRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, *lpuart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_LPUART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_DriverIRQHandler(uint32_t instance)

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- instance – LPUART instance.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually

Some flag can't manually clear

enumerator kStatus_LPUART_Error

Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun

LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select
LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag
Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
LPUART transmit CTS configure.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *_lpuart_config* lpuart_config_t
LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_config
#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
RX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

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

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.

2.44 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

2.45 Mc_rgm

static inline uint32_t MC_RGM_GetDestuctiveResetSourcesStatus(MC_RGM_Type *base)

Get destructive reset sources status.

Parameters

- base – MC_RGM peripheral base address.

Returns

Destructive reset sources status, This is the logical OR of members of _mc_rgm_destructive_reset_sources_flag.

static inline void MC_RGM_ClearDestuctiveResetSourcesStatus(MC_RGM_Type *base, uint32_t flag)

Clear destructive reset sources status.

Parameters

- base – MC_RGM peripheral base address.
- flag – Destructive reset sources flag, it can be logical OR of members of _mc_rgm_destructive_reset_sources_flag.

static inline uint32_t MC_RGM_GetFunctionalResetSourcesStatus(MC_RGM_Type *base)

Get functional reset sources status.

Parameters

- base – MC_RGM peripheral base address.

Returns

Functional reset sources status, This is the logical OR of members of _mc_rgm_functional_reset_sources_flag.

static inline void MC_RGM_ClearFunctionalResetSourcesStatus(MC_RGM_Type *base, uint32_t flag)

Clear functional reset sources status.

Parameters

- base – MC_RGM peripheral base address.
- flag – Functional reset sources flag, it can be logical OR of members of _mc_rgm_functional_reset_sources_flag.

static inline uint32_t MC_RGM_GetResetDuringStandbyStatus(MC_RGM_Type *base)

Get reset during standby status.

Parameters

- base – MC_RGM peripheral base address.

Returns

Reset during standby status, This is the logical OR of members of _mc_rgm_reset_during_standby_status.

static inline void MC_RGM_ClearResetDuringStandbyStatus(MC_RGM_Type *base, uint32_t flag)

Clear reset during standby status.

Parameters

- base – MC_RGM peripheral base address.
- flag – Reset during standby status, it can be logical OR of members of _mc_rgm_reset_during_standby_status.

static inline void MC_RGM_DisableBidirectionalReset(MC_RGM_Type *base, uint32_t flag)

External reset pin is not asserted on a given 'functional' reset event.

Parameters

- base – MC_RGM peripheral base address.
- flag – Functional reset event, it can be logical OR of members of _mc_rgm_bidirectional_function_reset_sources.

static inline void MC_RGM_DemoteFunctionalResetToInterrupt(MC_RGM_Type *base, uint32_t mask)

Demotes selected functional reset sources to interrupts.

Parameters

- base – MC_RGM peripheral base address.
- mask – Functional reset sources to be demoted to interrupts, it can be logical OR of members of _mc_rgm_demotable_functional_reset_sources.

static inline void MC_RGM_SetDestructiveResetEscalationThreshold(MC_RGM_Type *base, uint32_t count)

Set the threshold for destructive reset escalation.

This function sets the threshold for destructive reset escalation. MC_RGM increases a counter on each destructive reset When the counter reaches the threshold, MC_RGM enters reset DEST0 and stays there until the next power-on reset occurs.

Note: This counter is cleared on a write of any value to the RGM_DRET register (call this function) and on any power-on reset.

Parameters

- base – MC_RGM peripheral base address.
- count – The threshold for destructive reset escalation.

static inline void MC_RGM_SetFunctionalResetEscalationThreshold(MC_RGM_Type *base, uint32_t count)

Set the threshold for functional reset escalation.

This function sets the threshold for functional reset escalation. MC_RGM increases a counter on each functional reset. When the counter reaches the threshold, MC_RGM asserts a destructive reset.

Note: This counter is cleared on a write of any value to the RGM_FRET register (call this function) and on any power-on or destructive reset.

Parameters

- base – MC_RGM peripheral base address.
- count – The threshold for functional reset escalation.

static inline uint32_t MC_RGM_GetFunctionalResetEscalationCount(MC_RGM_Type *base)

Get the current count of functional reset escalation counter.

Parameters

- base – MC_RGM peripheral base address.

Returns
The current count of functional reset escalation counter.

FSL_MC_RGM_DRIVER_VERSION
MC_RGM driver version 2.1.0.

enum _mc_rgm_destructive_reset_sources_flag
Destructive reset sources flag.

Values:

enumerator kMC_RGM_PowerOnResetFlag
A power-on destructive reset (POR) has occurred

enumerator kMC_RGM_FccuFailureToReactResetFlag
FCCU failure to react destructive reset (FCCU_FTR) has occurred

enumerator kMC_RGM_StcuUnrecoverableFaultResetFlag
STCU unrecoverable fault (STCU_URF) destructive reset has occurred

enumerator kMC_RGM_FunctionalResetEscalation
Functional reset escalation (MCRGM_FRE) destructive reset has occurred

enumerator kMC_RGM_FxoscFailureResetFlag
FXOSC failure (FXOSC_FAIL) destructive reset has occurred

enumerator kMC_RGM_PllLossOfLockResetFlag
PLL loss of lock (PLL_LOL) destructive reset has occurred

enumerator kMC_RGM_CoreClockFailureResetFlag
Core clock failure (CORE_CLK_FAIL) destructive reset has occurred

enumerator kMC_RGM_AipsPlatClockFailureResetFlag
AIPS_PLAT_CLK failure (HSE_CLK_FAIL) destructive reset has occurred

enumerator kMC_RGM_HseClockFailureResetFlag
HSE_CLK failure (SYS_DIV_FAIL) destructive reset has occurred

enumerator kMC_RGM_SystemDividerFailureResetFlag
System clock dividers alignment failure reset (SYS_DIV_FAIL) has occurred.

enumerator kMC_RGM_HseTamperDetectResetFlag
HSE_B tamper detection reset (HSE_TMPR_RST) has occurred.

enumerator kMC_RGM_HseSnvsTamperDetectionResetFlag
HSE_B SNVS tamper detection reset (HSE_SNVS_RST) has occurred.

enumerator kMC_RGM_SoftwareDestructiveResetFlag
Software destructive reset (SW_DEST) has occurred.

enumerator kMC_RGM_DebugDestructiveResetFlag
Debug destructive reset (DEBUG_DEST) has occurred.

enumerator kMC_RGM_AllDestructiveResetFlags

enum _mc_rgm_functional_reset_sources_flag
Functional reset sources flag.

Values:

enumerator kMC_RGM_ExternalDestructiveResetFLag
An external destructive reset (EXR) has occurred

enumerator kMC_RGM_FccuReactionResetFlag
FCCU reaction (FCCU_RST) functional reset has occurred

enumerator kMC_RGM_SelfTestDoneDoneFlag
Self-test done (STCU_DONE) functional reset has occurred

enumerator kMC_RGM_Swt0Reset0Flag
SWT0 functional reset (SWT0_RST) has occurred

enumerator kMC_RGM_JtagResetFlag
JTAG functional reset (JTAG_RST) has occurred

enumerator kMC_RGM_HseSoftwareTriggerResetFlag
HSE_B SWT functional reset (HSE_SWT_RST) has occurred

enumerator kMC_RGM_HseBootResetFlag
HSE_B boot functional reset (HSE_BOOT_RST) has occurred

enumerator kMC_RGM_SoftwareFunctionalResetFlag
Software functional reset (SW_FUNC) has occurred

enumerator kMC_RGM_DebugFunctionalResetFlag
Debug functional reset (DEBUG_FUNC) has occurred

enumerator kMC_RGM_AllFunctionalResetFlags

enum _mc_rgm_bidirectional_function_reset_sources
Bidirectional functional reset sources.

Values:

enumerator kMC_RGM_BidirectionalDebugFunctionalReset
External reset pin is not asserted on a 'functional' reset event DEBUG_FUNC

enumerator kMC_RGM_BidirectionalSoftwareFunctionalReset
External reset pin is not asserted on a 'functional' reset event SW_FUNC

enumerator kMC_RGM_BidirectionalHseBootReset
External reset pin is not asserted on a 'functional' reset event HSE_BOOT_RST

enumerator kMC_RGM_BidirectionalHseSwtReset
External reset pin is not asserted on a 'functional' reset event HSE_SWT_RST

enumerator kMC_RGM_BidirectionalJtagReset
External reset pin is not asserted on a 'functional' reset event JTAG_RST

enumerator kMC_RGM_BidirectionalSwt0Reset
External reset pin is not asserted on a 'functional' reset event SWT0_RST

enumerator kMC_RGM_BidirectionalSelfTestDoneReset
External reset pin is not asserted on a 'functional' reset event ST_DONE

enumerator kMC_RGM_BidirectionalFccuReactionReset
External reset pin is not asserted on a 'functional' reset event FCCU_RST

enum _mc_rgm_demotable_functional_reset_sources
Demotable functional reset sources.

Values:

enumerator kMC_RGM_FccuReactionReset
Functional reset event FCCU_RST generates an interrupt request.

enumerator kMC_RGM_Swt0Reset
Functional reset event SWT0_RST generates an interrupt request.

enumerator kMC_RGM_JtagReset
Functional reset event DEBUG_FUNC generates an interrupt request.

enumerator kMC_RGM_DebugFunctionalReset
Functional reset event DEBUG_FUNC generates an interrupt request.

enum _mc_rgm_reset_during_standby_status
Reset during standby status.

Values:

enumerator kMC_RGM_DestructiveResetDuringStandby
Destructive reset event occurred during standby mode.

enumerator kMC_RGM_FunctionalResetDuringStandby
Functional reset event occurred during standby mode.

2.46 MCM: Miscellaneous Control Module

FSL_MCM_DRIVER_VERSION
MCM driver version.

Enum _mcm_interrupt_flag. Interrupt status flag mask. .

Values:

enumerator kMCM_CacheWriteBuffer
Cache Write Buffer Error Enable.

enumerator kMCM_ParityError
Cache Parity Error Enable.

enumerator kMCM_FPUInvalidOperation
FPU Invalid Operation Interrupt Enable.

enumerator kMCM_FPUDivideByZero
FPU Divide-by-zero Interrupt Enable.

enumerator kMCM_FPUOverflow
FPU Overflow Interrupt Enable.

enumerator kMCM_FPUUnderflow
FPU Underflow Interrupt Enable.

enumerator kMCM_FPUInexact
FPU Inexact Interrupt Enable.

enumerator kMCM_FPUInputDenormalInterrupt
FPU Input Denormal Interrupt Enable.

typedef union *_mcm_buffer_fault_attribute* mcm_buffer_fault_attribute_t
The union of buffer fault attribute.

typedef union *_mcm_lmem_fault_attribute* mcm_lmem_fault_attribute_t
The union of LMEM fault attribute.

static inline void MCM_EnableCrossbarRoundRobin(MCM_Type *base, bool enable)

Enables/Disables crossbar round robin.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
  - **true** Enable crossbar round robin.
  - **false** disable crossbar round robin.

static inline void MCM_EnableInterruptStatus(MCM_Type *base, uint32_t mask)

Enables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(_mcm_interrupt_flag).

static inline void MCM_DisableInterruptStatus(MCM_Type *base, uint32_t mask)

Disables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(_mcm_interrupt_flag).

static inline uint16_t MCM_GetInterruptStatus(MCM_Type *base)

Gets the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline void MCM_ClearCacheWriteBufferErroStatus(MCM_Type *base)

Clears the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultAddress(MCM_Type *base)

Gets buffer fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetBufferFaultAttribute(MCM_Type *base, *mcm_buffer_fault_attribute_t* *bufferfault)

Gets buffer fault attributes.

Parameters

- base – MCM peripheral base address.
- bufferfault – Structure to store the result.

static inline uint32_t MCM_GetBufferFaultData(MCM_Type *base)

Gets buffer fault data.

Parameters

- base – MCM peripheral base address.

static inline void MCM_LimitCodeCachePeripheralWriteBuffering(MCM_Type *base, bool enable)
Limit code cache peripheral write buffering.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
  - **true** Enable limit code cache peripheral write buffering.
  - **false** disable limit code cache peripheral write buffering.

static inline void MCM_BypassFixedCodeCacheMap(MCM_Type *base, bool enable)
Bypass fixed code cache map.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
  - **true** Enable bypass fixed code cache map.
  - **false** disable bypass fixed code cache map.

static inline void MCM_EnableCodeBusCache(MCM_Type *base, bool enable)
Enables/Disables code bus cache.

Parameters

- base – MCM peripheral base address.
- enable – Used to disable/enable code bus cache.
  - **true** Enable code bus cache.
  - **false** disable code bus cache.

static inline void MCM_ForceCodeCacheToNoAllocation(MCM_Type *base, bool enable)
Force code cache to no allocation.

Parameters

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
  - **true** Force code cache to no allocation.
  - **false** Force code cache to allocation.

static inline void MCM_EnableCodeCacheWriteBuffer(MCM_Type *base, bool enable)
Enables/Disables code cache write buffer.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
  - **true** Enable code cache write buffer.
  - **false** Disable code cache write buffer.

static inline void MCM_ClearCodeBusCache(MCM_Type *base)
Clear code bus cache.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnablePcParityFaultReport(MCM_Type *base, bool enable)

Enables/Disables PC Parity Fault Report.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
  - **true** Enable PC Parity Fault Report.
  - **false** disable PC Parity Fault Report.

static inline void MCM_EnablePcParity(MCM_Type *base, bool enable)

Enables/Disables PC Parity.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
  - **true** Enable PC Parity.
  - **false** disable PC Parity.

static inline void MCM_LockConfigState(MCM_Type *base)

Lock the configuration state.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnableCacheParityReporting(MCM_Type *base, bool enable)

Enables/Disables cache parity reporting.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
  - **true** Enable cache parity reporting.
  - **false** disable cache parity reporting.

static inline uint32_t MCM_GetLmemFaultAddress(MCM_Type *base)

Gets LMEM fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetLmemFaultAttribute(MCM_Type *base, *mcm_lmem_fault_attribute_t* *lmemFault)

Get LMEM fault attributes.

Parameters

- base – MCM peripheral base address.
- lmemFault – Structure to store the result.

static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)

Gets LMEM fault data.

Parameters

- base – MCM peripheral base address.

MCM_LMFATR_TYPE_MASK

MCM_LMFATR_MODE_MASK

MCM_LMFATR_BUFF_MASK

MCM_LMFATR_CACH_MASK

MCM_ISCR_STAT_MASK

FSL_COMPONENT_ID

union _mcm_buffer_fault_attribute

#include <fsl_mcm.h> The union of buffer fault attribute.

Public Members

uint32_t attribute

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

struct *_mcm_buffer_fault_attribute._mcm_buffer_fault_attribut* attribute_memory

struct _mcm_buffer_fault_attribut

#include <fsl_mcm.h>

Public Members

uint32_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32_t busErrorSize

Indicates the size of the cache write buffer access.

uint32_t busErrorAccess

Indicates the type of system bus access.

uint32_t busErrorMasterID

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32_t busErrorOverrun

Indicates if another cache write buffer bus error is detected.

union _mcm_lmem_fault_attribute

#include <fsl_mcm.h> The union of LMEM fault attribute.

Public Members

uint32_t attribute

Indicates the attributes of the LMEM fault detected.

struct *_mcm_lmem_fault_attribute._mcm_lmem_fault_attribut* attribute_memory

struct _mcm_lmem_fault_attribut

#include <fsl_mcm.h>

Public Members

uint32_t parityFaultProtectionSignal
Indicates the features of parity fault protection signal.

uint32_t parityFaultMasterSize
Indicates the parity fault master size.

uint32_t parityFaultWrite
Indicates the parity fault is caused by read or write.

uint32_t backdoorAccess
Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32_t parityFaultSyndrome
Indicates the parity fault syndrome.

uint32_t overrun
Indicates the number of faultss.

2.47 MSCM: Miscellaneous System Control

FSL_MSCM_DRIVER_VERSION
MSCM driver version 2.0.0.

typedef struct *_mscm_uid* mscm_uid_t

static inline void MSCM_GetUID(MSCM_Type *base, *mscm_uid_t* *uid)
Get MSCM UID.

Parameters

- base – MSCM peripheral base address.
- uid – Pointer to an uid struct.

static inline void MSCM_SetSecureIrqParameter(MSCM_Type *base, const uint32_t parameter)
Set MSCM Secure Irq.

Parameters

- base – MSCM peripheral base address.
- parameter – Value to be write to SECURE_IRQ.

static inline uint32_t MSCM_GetSecureIrq(MSCM_Type *base)
Get MSCM Secure Irq.

Parameters

- base – MSCM peripheral base address.

Returns
MSCM Secure Irq.

FSL_COMPONENT_ID

struct _mscm_uid
#include <fsl_mscm.h>

2.48 PIT: Periodic Interrupt Timer

void PIT_Init(PIT_Type *base, const *pit_config_t* *config)

Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.

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

Parameters

- base – PIT peripheral base address
- config – Pointer to the user's PIT config structure

void PIT_Deinit(PIT_Type *base)

Gates the PIT clock and disables the PIT module.

Parameters

- base – PIT peripheral base address

void PIT_RTI_Init(PIT_Type *base, const *pit_config_t* *config)

Enables the PIT RTI module, and configures the peripheral for basic operations.

Note: The RTI might take several RTI clock cycles to get enabled or updated. Hence, you must wait for at least four RTI clock cycles after RTI configuration.

Parameters

- base – PIT peripheral base address
- config – Pointer to the user's PIT config structure

static inline void PIT_RTI_Deinit(PIT_Type *base)

Disables the PIT RTI module.

Parameters

- base – PIT peripheral base address

static inline void PIT_GetDefaultConfig(*pit_config_t* *config)

Fills in the PIT configuration structure with the default settings.

The default values are as follows.

```
config->enableRunInDebug = false;
```

Parameters

- config – Pointer to the configuration structure.

static inline void PIT_SetTimerChainMode(PIT_Type *base, *pit_chnl_t* channel, bool enable)

Enables or disables chaining a timer with the previous timer.

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

- base – PIT peripheral base address

- channel – Timer channel number which is chained with the previous timer
- enable – Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers.

static inline void PIT_EnableInterrupts(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Enables the selected PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration pit_interrupt_enable_t

static inline void PIT_DisableInterrupts(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Disables the selected PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The interrupts to disable. This is a logical OR of members of the enumeration pit_interrupt_enable_t

static inline uint32_t PIT_GetEnabledInterrupts(PIT_Type *base, *pit_chnl_t* channel)

Gets the enabled PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

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

static inline void PIT_EnableRtiInterrupts(PIT_Type *base, uint32_t mask)

Enables the PIT RTI interrupts.

Parameters

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

static inline void PIT_DisableRtiInterrupts(PIT_Type *base, uint32_t mask)

Disables the selected PIT RTI interrupts.

Parameters

- base – PIT peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration pit_rti_interrupt_enable_t

static inline uint32_t PIT_GetEnabledRtiInterrupts(PIT_Type *base)

Gets the enabled PIT RTI interrupts.

Parameters

- base – PIT peripheral base address

Returns
The enabled interrupts. This is the logical OR of members of the enumeration pit_rti_interrupt_enable_t

static inline uint32_t PIT_GetStatusFlags(PIT_Type *base, *pit_chnl_t* channel)

Gets the PIT status flags.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns
The status flags. This is the logical OR of members of the enumeration pit_status_flags_t

static inline void PIT_ClearStatusFlags(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Clears the PIT status flags.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration pit_status_flags_t

static inline uint32_t PIT_GetRtiStatusFlags(PIT_Type *base)

Gets the PIT RTI flags.

Parameters

- base – PIT peripheral base address

Returns
The status flags. This is the logical OR of members of the enumeration pit_rti_status_flags_t

static inline void PIT_ClearRtiStatusFlags(PIT_Type *base, uint32_t mask)

Clears the PIT RTI status flags.

Parameters

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

static inline uint32_t PIT_GetRtiSyncStatus(PIT_Type *base)

Reads the RTI timer load synchronization status.

In the case of the RTI timer load, it will take several cycles until this value is synchronized into the RTI clock domain.

Parameters

- base – PIT peripheral base address

Returns
The status flags. This is the logical OR of members of the enumeration pit_rti_ldval_status_flags_t

static inline void PIT_ClearRtiSyncStatus(PIT_Type *base)

Clears the RTI timer load synchronization status.

Parameters

- base – PIT peripheral base address

static inline void PIT_SetTimerPeriod(PIT_Type *base, *pit_chnl_t* channel, uint32_t count)

Sets the timer period in units of count.

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: Users can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- count – Timer period in units of ticks

static inline uint32_t PIT_GetCurrentTimerCount(PIT_Type *base, *pit_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: Users can call the utility macros provided in fsl_common.h to convert ticks to usec or msec.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

Current timer counting value in ticks

static inline void PIT_SetRtiTimerPeriod(PIT_Type *base, uint32_t count)

Sets the RTI timer period in units of count.

RTI timer begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: Users can call the utility macros provided in fsl_common.h to convert to ticks. it will take several cycles until this value is synchronized into the RTI clock domain. So, in user code, it is recommended to check the RTI_LDVAL_STAT register

```
PIT_ClearRtiSyncStatus(base);
PIT_SetRtiTimerPeriod(base, count);
while(kPIT_RtiLoadValueSyncFlag != (PIT_GetRtiSyncStatus(base)))
{
}
```

However, according to ERR050763, this is not reliable for dynamic load mode (User set a new counter period without restarting the timer). In such case, user shall manually check PIT_GetRtiTimerCount() to ensure the current timer expires and the new value was loaded.

Parameters

- base – PIT peripheral base address
- count – Timer period in units of ticks

static inline uint32_t PIT_GetRtiTimerCount(PIT_Type *base)

Reads the RTI timer counting value.

This function returns the real-time RTI timer counting value, in a range from 0 to a timer period.

Note: Users can call the utility macros provided in fsl_common.h to convert ticks to usec or msec.

Parameters

- base – PIT peripheral base address

Returns

RTI timer counting value in ticks

static inline void PIT_StartTimer(PIT_Type *base, *pit_chnl_t* channel)

Starts the timer counting.

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number.

static inline void PIT_StopTimer(PIT_Type *base, *pit_chnl_t* channel)

Stops the timer counting.

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number.

static inline void PIT_StartRtiTimer(PIT_Type *base)

Starts the RTI timer counting.

After calling this function, RTI timer load period value, count down to 0 and then load the respective start value again. Each time RTI timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- base – PIT peripheral base address

static inline void PIT_StopRtiTimer(PIT_Type *base)

Stops the RTI timer counting.

This function stops every RTI timer counting. RTI timer reloads its periods respectively after the next time it call the PIT_DRV_StartRtiTimer.

Parameters

- base – PIT peripheral base address

FSL_PIT_DRIVER_VERSION

PIT Driver Version 2.2.0.

enum _pit_chnl
List of PIT channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPIT_Chnl_0
PIT channel number 0

enumerator kPIT_Chnl_1
PIT channel number 1

enumerator kPIT_Chnl_2
PIT channel number 2

enumerator kPIT_Chnl_3
PIT channel number 3

enum _pit_interrupt_enable
List of PIT interrupts.

Values:

enumerator kPIT_TimerInterruptEnable
Timer interrupt enable

enum _pit_status_flags
List of PIT status flags.

Values:

enumerator kPIT_TimerFlag
Timer flag

enum _pit_rti_interrupt_enable
List of PIT RTI interrupts.

Values:

enumerator kPIT_RtiTimerInterruptEnable
Real time interrupt enable

enum _pit_rti_status_flags
List of PIT RTI status flags.

Values:

enumerator kPIT_RtiTimerFlag
Real time interrupt flag

enum _pit_rti_ldval_status_flags
List of PIT RTI timer load value sync status flags.

Values:

enumerator kPIT_RtiLoadValueSyncFlag

typedef enum *_pit_chnl* pit_chnl_t
List of PIT channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pit_interrupt_enable* pit_interrupt_enable_t

List of PIT interrupts.

typedef enum *_pit_status_flags* pit_status_flags_t

List of PIT status flags.

typedef enum *_pit_rti_interrupt_enable* pit_rti_interrupt_enable_t

List of PIT RTI interrupts.

typedef enum *_pit_rti_status_flags* pit_rti_status_flags_t

List of PIT RTI status flags.

typedef enum *_pit_rti_ldval_status_flags* pit_rti_ldval_status_flags_t

List of PIT RTI timer load value sync status flags.

typedef struct *_pit_config* pit_config_t

PIT configuration structure.

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

The configuration structure can be made constant so it resides in flash.

uint64_t PIT_GetLifetimeTimerCount(PIT_Type *base)

Reads the current lifetime counter value.

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the PIT_SetTimerChainMode before using this timer. The period of lifetime timer is equal to the "period of timer 0 * period of timer 1". For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

- base – PIT peripheral base address

Returns

Current lifetime timer value

struct _pit_config

#include <fsl_pit.h> PIT configuration structure.

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

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

2.49 PMC: Power Management Controller

2.50 Power

enum _pmc_event_flag

Power PMC voltage event flag enumeration.

Values:

enumerator kPMC_FPMVDD_HV_ALowPowerVoltageResetEventFlag

Low-voltage reset event on VDD_HV_A domain in FPM(full performance mode).

enumerator kPMC_LPMFPMVDD_HV_ALowPowerVoltageResetEventFlag

Low-power reset event on VDD_HV_A domain in both LPM(low performance mode) and FPM.

enumerator kPMC_FPMVDD_HV_BLowPowerVoltageResetEventFlag

Low-voltage reset event on VDD_HV_B domain in FPM.

enumerator kPMC_LPMFPMVDD_HV_BLowPowerVoltageResetEventFlag

Low-power reset event on VDD_HV_B domain in both LPM and FPM.

enumerator kPMC_FPMV25LowPowerVoltageResetEventFlag

Low-voltage reset event on V25 domain in FPM.

enumerator kPMC_LPMFPMV25LowPowerVoltageResetEventFlag

Low-power reset event on V25 domain in both LPM and FPM.

enumerator kPMC_FPMV11LowPowerVoltageResetEventFlag

Low-voltage reset event on V11 1.1V domain in FPM.

enumerator kPMC_LPMFPMV11LowPowerVoltageResetEventFlag

Low-power reset event on V11 1.1V domain in both LPM and FPM.

enumerator kPMC_FPMOSCV25GoNoGoEventFlag

Go/NoGo sensor has detected a low voltage on the V25 domain in FPM.

enumerator kPMC_FPMOSCV11GoNoGoEventFlag

Go/NoGo sensor has detected a low voltage on the V11 domain in FPM.

enumerator kPMC_POREventFlag

POR event flag.

enumerator kPMC_AllEventFlags

All event flags.

enum _pmc_voltage_detect_flag

List of Power PMC high and low voltage detect flags enumeration.

Values:

enumerator kPMC_FPMVDD_HV_AHighVoltageDetectFlag

Voltage on VDD_HV_A is above high-voltage detect threshold and FPM is active.

enumerator kPMC_FPMVDD_HV_BHighVoltageDetectFlag

Voltage on VDD_HV_B is above high-voltage detect threshold and FPM is active.

enumerator kPMC_FPMV25HighVoltageDetectFlag

Voltage on V25 is above high-voltage detect threshold and full performance mode is active.

enumerator kPMC_FPMV11HighVoltageDetectFlag

Voltage on V11 is above high-voltage detect threshold and full performance mode is active.

enumerator kPMC_FPMAllHighVoltageDetectFlags

All high voltage detect flags.

enumerator kPMC_VDD_HV_AV5LowVoltageDetectFlag
Voltage on VDD_HV_A is below 5V low-voltage threshold.

enumerator kPMC_V15LowVoltageDetectFlag
Voltage on V15 is below low-voltage detect threshold and full performance mode is active.

enumerator kPMC_AllLowVoltageDetectFlags
All low voltage detect flags.

enumerator kPMC_AllVoltageDetectFlags
All voltage detect flags.

enum _pmc_voltage_detect_status
List of POWER PMC high and low voltage detect status enumeration.

Values:

enumerator kPMC_FPMVDD_HV_AHighVoltageStatus
Voltage on VDD_HV_A is above high-voltage detect threshold and FPM is active.

enumerator kPMC_FPMVDD_HV_BHighVoltageStatus
Voltage on VDD_HV_B is above high-voltage detect threshold and FPM is active.

enumerator kPMC_FPMV25HighVoltageStatus
Voltage on V25 is above high-voltage detect threshold and FPM is active.

enumerator kPMC_FPMV11HighVoltageStatus
Voltage on V11 is above high-voltage detect threshold and FPM is active.

enumerator kPMC_VDD_HV_AV5LowVoltageStatus
Voltage on VDD_HV_A is below low-voltage detect threshold.

enumerator kPMC_V15LowVoltageStatus
Voltage on V15 is below low-voltage detect threshold and FPM is active.

enumerator kPMC_AllVoltageDetectStatus
All voltage detect status.

enum _pmc_interrupt_enable
List of PMC interrupts.

Values:

enumerator kPMC_HighVoltageDetectInterruptEnable
High voltage detect interrupt enable.

enumerator kPMC_LowVoltageDetectInterruptEnable
Low voltage detect interrupt enable.

enumerator kPMC_AnyVoltageDetectInterruptEnable
Hign and low voltage detect interrupt enable.

typedef enum *_pmc_event_flag* pmc_event_flag_t
Power PMC voltage event flag enumeration.

typedef enum *_pmc_voltage_detect_flag* pmc_voltage_detect_flag_t
List of Power PMC high and low voltage detect flags enumeration.

typedef enum *_pmc_voltage_detect_status* pmc_voltage_detect_status_t
List of POWER PMC high and low voltage detect status enumeration.

typedef enum *_pmc_interrupt_enable* pmc_interrupt_enable_t
List of PMC interrupts.

typedef struct *_pmc_version_id* pmc_version_id_t
POWER PMC IP version ID definition.

enum _mc_me_previous_mode
POWER MC_ME previous mode enumeration.

Values:

enumerator kMC_ME_PreviousResetMode
Previous mode is reset mode(any reset).

enumerator kMC_ME_PreviousStandbyMode
Previous mode is standby mode.

enum _mc_me_reset_type
POWER MC_ME reset type.

Values:

enumerator kMC_ME_DestructiveReset
Destructive reset request

enumerator kMC_ME_FunctionalReset
Functional reset request

typedef enum *_mc_me_reset_type* mc_me_reset_type_t
POWER MC_ME reset type.

static inline void POWER_EnableLastMileRegulator(bool externalBJTUsed)
Enable POWER PMC last mile regulator.

Parameters

- externalBJTUsed – Set to true if external BJT between VDD_HV_A and V15 is used, otherwise set to false

static inline void POWER_DisableLastMileRegulator(void)
Disables POWER PMC last mile regulator.

static inline void POWER_EnableFastRecoveryFromLPM(void)
Enable fast recovery from LPM.

static inline void POWER_DisableFastRecoveryFromLPM(void)
Disable fast recovery from LPM.

static inline void POWER_EnableLPMV25Regulator(void)
Enable V25 domain regulator.

static inline void POWER_DisableLPMV25Regulator(void)
Disable V25 domain regulator.

static inline void POWER_EnableLPMVDD_HV_BLowVoltageReset(void)
Enable low voltage reset in VDD_HV_B domain of LPM.

static inline void POWER_DisableLPMVDD_HV_BLowVoltageReset(void)
Disable low voltage reset in VDD_HV_B domain of LPM.

static inline void POWER_EnableAutoTurnOverLastMileRegulator(void)
Enable auto turn over last mile regulator.

static inline void POWER_DisableAutoTurnOverLastMileRegulator(void)
Disable auto turn over last mile regulator.

static inline uint32_t POWER_GetPMCEventFlags(void)

Get the PMC event flag.

Returns

The event flags. This is the logical OR of members of the enumeration pmc_event_flag_t

static inline void POWER_ClearPMCEventFlags(uint32_t mask)

Clear the event flag.

Parameters

- mask – The event flags to clear. This is a logical OR of members of the enumeration pmc_event_flag_t

static inline uint32_t POWER_GetVoltageDetectStatus(void)

Get the voltage detect status.

Returns

The voltage detect status. This is the logical OR of members of the enumeration pmc_voltage_detect_status_t

static inline void POWER_EnablePMCInterrupts(uint32_t mask)

Enable POWER PMC voltage detect interrupt.

Parameters

- mask – The interrupts to enable. This is a logical OR of members of the enumeration pmc_interrupt_enable_t

static inline void POWER_DisablePMCInterrupts(uint32_t mask)

Disable POWER PMC voltage detect interrupt.

Parameters

- mask – The interrupts to disable. This is a logical OR of members of the enumeration pmc_interrupt_enable_t

static inline uint32_t POWER_GetPMCInterruptFlags(void)

Get PMC interrupt flags.

Returns

The voltage interrupt flags. This is the logical OR of members of the enumeration pmc_voltage_detect_flag_t

static inline void POWER_ClearPMCInterruptFlags(uint32_t mask)

Clear PMC interrupt flags.

Parameters

- mask – The interrupts to clear. This is a logical OR of members of the enumeration pmc_voltage_detect_flag_t

void POWER_GetPMCVersionID(*pmc_version_id_t* *versionId)

Gets the POWER PMC version id.

This function gets the PMC version id including whether the PMC supports last mile regulator.

Parameters

- versionId – Pointer to pmc_version_id_t structure.

static inline bool POWER_ExitFromStandbyMode(void)

Check previous mode is standby mode or not.

Returns

true if previous mode is standby mode, false if previous mode is reset mode.

void POWER_EnterStandbyMode(void)

Request to enter standby mode.

void POWER_RequestResetMode(*mc_me_reset_type_t* resetType)

Request destructive or functional reset mode transition.

Parameters

- resetType – Reset type, see ref mc_me_reset_type_t.

FSL_POWER_DRIVER_VERSION

POWER driver version.

Version 2.0.0. \

typedef void (*power_pmc_callback_t)(uint32_t mask)

POWER PMC callback function.

struct _pmc_version_id

#include <fsl_power.h> POWER PMC IP version ID definition.

Public Members

bool supportLastMileRegulator

Support last mile regulator feature.

uint8_t minor

Minor version number.

uint8_t major

Major version number.

2.51 QSPI: Quad Serial Peripheral Interface

2.52 Quad Serial Peripheral Interface Driver

uint32_t QSPI_GetInstance(QuadSPI_Type *base)

Get the instance number for QSPI.

Parameters

- base – QSPI base pointer.

void QSPI_Init(QuadSPI_Type *base, *qspi_config_t* *config, uint32_t srcClock_Hz)

Initializes the QSPI module and internal state.

This function enables the clock for QSPI and also configures the QSPI with the input configure parameters. Users should call this function before any QSPI operations.

Parameters

- base – Pointer to QuadSPI Type.
- config – QSPI configure structure.
- srcClock_Hz – QSPI source clock frequency in Hz.

void QSPI_GetDefaultQspiConfig(*qspi_config_t* *config)

Gets default settings for QSPI.

Parameters

- config – QSPI configuration structure.

void QSPI_Deinit(QuadSPI_Type *base)

Deinitializes the QSPI module.

Clears the QSPI state and QSPI module registers.

Parameters

- base – Pointer to QuadSPI Type.

void QSPI_SetFlashConfig(QuadSPI_Type *base, *qspi_flash_config_t* *config)

Configures the serial flash parameter.

This function configures the serial flash 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 QSPI features.

Parameters

- base – Pointer to QuadSPI Type.
- config – Flash configuration parameters.

void QSPI_SetDelayChainConfig(QuadSPI_Type *base, *qspi_delay_chain_config_t* *config)

Configures the delay chain parameter.

This function configures the slave delay chain.

Parameters

- base – Pointer to QuadSPI Type.
- config – Delay chain configuration parameters.

void QSPI_SoftwareReset(QuadSPI_Type *base)

Software reset for the QSPI 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 – Pointer to QuadSPI Type.

static inline void QSPI_Enable(QuadSPI_Type *base, bool enable)

Enables or disables the QSPI module.

Parameters

- base – Pointer to QuadSPI Type.
- enable – True means enable QSPI, false means disable.

static inline uint32_t QSPI_GetStatusFlags(QuadSPI_Type *base)

Gets the state value of QSPI.

Parameters

- base – Pointer to QuadSPI Type.

Returns

status flag, use status flag to AND _qspi_flags could get the related status.

static inline uint32_t QSPI_GetErrorStatusFlags(QuadSPI_Type *base)

Gets QSPI error status flags.

Parameters

- base – Pointer to QuadSPI Type.

Returns

status flag, use status flag to AND _qspi_error_flags could get the related status.

static inline void QSPI_ClearErrorFlag(QuadSPI_Type *base, uint32_t mask)

Clears the QSPI error flags.

Parameters

- base – Pointer to QuadSPI Type.
- mask – Which kind of QSPI flags to be cleared, a combination of _qspi_error_flags.

static inline void QSPI_EnableInterrupts(QuadSPI_Type *base, uint32_t mask)

Enables the QSPI interrupts.

Parameters

- base – Pointer to QuadSPI Type.
- mask – QSPI interrupt source.

static inline void QSPI_DisableInterrupts(QuadSPI_Type *base, uint32_t mask)

Disables the QSPI interrupts.

Parameters

- base – Pointer to QuadSPI Type.
- mask – QSPI interrupt source.

static inline void QSPI_EnableDMA(QuadSPI_Type *base, uint32_t mask, bool enable)

Enables the QSPI DMA source.

Parameters

- base – Pointer to QuadSPI Type.
- mask – QSPI DMA source.
- enable – True means enable DMA, false means disable.

static inline uint32_t QSPI_GetTxDataRegisterAddress(QuadSPI_Type *base)

Gets the Tx data register address. It is used for DMA operation.

Parameters

- base – Pointer to QuadSPI Type.

Returns

QSPI Tx data register address.

uint32_t QSPI_GetRxDataRegisterAddress(QuadSPI_Type *base)

Gets the Rx data register address used for DMA operation.

This function returns the Rx data register address or Rx buffer address according to the Rx read area settings.

Parameters

- base – Pointer to QuadSPI Type.

Returns

QSPI Rx data register address.

static inline void QSPI_SetIPCommandAddress(QuadSPI_Type *base, uint32_t addr)

Sets the IP command address.

Parameters

- base – Pointer to QuadSPI Type.
- addr – IP command address.

static inline void QSPI_SetIPCommandSize(QuadSPI_Type *base, uint32_t size)

Sets the IP command size.

Parameters

- base – Pointer to QuadSPI Type.
- size – IP command size.

void QSPI_ExecuteIPCommand(QuadSPI_Type *base, uint32_t index)

Executes IP commands located in LUT table.

Parameters

- base – Pointer to QuadSPI Type.
- index – IP command located in which LUT table index.

void QSPI_ExecuteAHBCommand(QuadSPI_Type *base, uint32_t index)

Executes AHB commands located in LUT table.

Parameters

- base – Pointer to QuadSPI Type.
- index – AHB command located in which LUT table index.

static inline void QSPI_EnableIPParallelMode(QuadSPI_Type *base, bool enable)

Enables/disables the QSPI IP command parallel mode.

Parameters

- base – Pointer to QuadSPI Type.
- enable – True means enable parallel mode, false means disable parallel mode.

static inline void QSPI_EnableAHBParallelMode(QuadSPI_Type *base, bool enable)

Enables/disables the QSPI AHB command parallel mode.

Parameters

- base – Pointer to QuadSPI Type.
- enable – True means enable parallel mode, false means disable parallel mode.

void QSPI_UpdateLUT(QuadSPI_Type *base, uint32_t index, uint32_t *cmd)

Updates the LUT table.

Parameters

- base – Pointer to QuadSPI Type.
- index – Which LUT index needs to be located. It should be an integer divided by 4.
- cmd – Command sequence array.

static inline void QSPI_ClearFifo(QuadSPI_Type *base, uint32_t mask)

Clears the QSPI FIFO logic.

Parameters

- base – Pointer to QuadSPI Type.
- mask – Which kind of QSPI FIFO to be cleared.

static inline void QSPI_ClearCommandSequence(QuadSPI_Type *base, *qspi_command_seq_t* seq)

@ brief Clears the command sequence for the IP/buffer command.

This function can reset the command sequence.

Parameters

- base – QSPI base address.
- seq – Which command sequence need to reset, IP command, buffer command or both.

static inline void QSPI_EnableDDRMode(QuadSPI_Type *base, bool enable)

Enable or disable DDR mode.

Parameters

- base – QSPI base pointer
- enable – True means enable DDR mode, false means disable DDR mode.

void QSPI_SetReadDataArea(QuadSPI_Type *base, *qspi_read_area_t* area)

@ brief Set the RX buffer readout area.

This function can set the RX buffer readout, from AHB bus or IP Bus.

Parameters

- base – QSPI base address.
- area – QSPI Rx buffer readout area. AHB bus buffer or IP bus buffer.

void QSPI_WriteBlocking(QuadSPI_Type *base, const uint32_t *buffer, size_t size)

Sends a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – QSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to send

static inline void QSPI_WriteData(QuadSPI_Type *base, uint32_t data)

Writes data into FIFO.

Parameters

- base – QSPI base pointer
- data – The data bytes to send

void QSPI_ReadBlocking(QuadSPI_Type *base, uint32_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. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

- base – QSPI base pointer
- buffer – The data bytes to send
- size – The number of data bytes to receive

uint32_t QSPI_ReadData(QuadSPI_Type *base)

Receives data from data FIFO.

Parameters

- base – QSPI base pointer

Returns

The data in the FIFO.

static inline void QSPI_TransferSendBlocking(QuadSPI_Type *base, *qspi_transfer_t* *xfer)

Writes data to the QSPI transmit buffer.

This function writes a continuous data to the QSPI transmit FIFO. This function is a block function and can return only when finished. This function uses polling methods.

Parameters

- base – Pointer to QuadSPI Type.
- xfer – QSPI transfer structure.

static inline void QSPI_TransferReceiveBlocking(QuadSPI_Type *base, *qspi_transfer_t* *xfer)

Reads data from the QSPI receive buffer in polling way.

This function reads continuous data from the QSPI receive buffer/FIFO. This function is a blocking function and can return only when finished. This function uses polling methods. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

- base – Pointer to QuadSPI Type.
- xfer – QSPI transfer structure.

FSL_QSPI_DRIVER_VERSION

QSPI driver version.

Status structure of QSPI.

Values:

enumerator kStatus_QSPI_Idle

QSPI is in idle state

enumerator kStatus_QSPI_Busy

QSPI is busy

enumerator kStatus_QSPI_Error
Error occurred during QSPI transfer

enum _qspi_read_area
QSPI read data area, from IP FIFO or AHB buffer.

Values:

enumerator kQSPI_ReadAHB
QSPI read from AHB buffer.

enumerator kQSPI_ReadIP
QSPI read from IP FIFO.

enum _qspi_command_seq
QSPI command sequence type.

Values:

enumerator kQSPI_IPSeq
IP command sequence

enumerator kQSPI_BufferSeq
Buffer command sequence

enumerator kQSPI_AllSeq

enum _qspi_fifo
QSPI buffer type.

Values:

enumerator kQSPI_TxFifo
QSPI Tx FIFO

enumerator kQSPI_RxFifo
QSPI Rx FIFO

enumerator kQSPI_AllFifo
QSPI all FIFO, including Tx and Rx

enum _qspi_error_flags
QSPI error flags.

Values:

enumerator kQSPI_DataLearningFail
Data learning pattern failure flag

enumerator kQSPI_TxBufferFill
Tx buffer fill flag

enumerator kQSPI_TxBufferUnderrun
Tx buffer underrun flag

enumerator kQSPI_IllegalInstruction
Illegal instruction error flag

enumerator kQSPI_RxBufferOverflow
Rx buffer overflow flag

enumerator kQSPI_RxBufferDrain
Rx buffer drain flag

enumerator kQSPI_AHBSequenceError
AHB sequence error flag

enumerator kQSPI_AHBBufferOverflow
AHB buffer overflow flag

enumerator kQSPI_IPCommandUsageError
IP command usage error flag

enumerator kQSPI_IPCommandTriggerDuringAHBAccess
IP command trigger during AHB access error

enumerator kQSPI_IPCommandTriggerDuringIPAccess
IP command trigger cannot be executed

enumerator kQSPI_IPCommandTriggerDuringAHBGrant
IP command trigger during AHB grant error

enumerator kQSPI_IPCommandTransactionFinished
IP command transaction finished flag

enumerator kQSPI_FlagAll
All error flag

enum _qspi_flags
QSPI state bit.

Values:

enumerator kQSPI_DataLearningSamplePoint
Data learning sample point

enumerator kQSPI_TxBufferFull
Tx buffer full flag

enumerator kQSPI_TxBufferEnoughData
Tx buffer enough data available

enumerator kQSPI_RxDMA
Rx DMA is requesting or running

enumerator kQSPI_RxBufferFull
Rx buffer full

enumerator kQSPI_RxWatermark
Rx buffer watermark exceeded

enumerator kQSPI_AHB3BufferFull
AHB buffer 3 full

enumerator kQSPI_AHB2BufferFull
AHB buffer 2 full

enumerator kQSPI_AHB1BufferFull
AHB buffer 1 full

enumerator kQSPI_AHB0BufferFull
AHB buffer 0 full

enumerator kQSPI_AHB3BufferNotEmpty
AHB buffer 3 not empty

enumerator kQSPI_AHB2BufferNotEmpty
AHB buffer 2 not empty

enumerator kQSPI_AHB1BufferNotEmpty
AHB buffer 1 not empty

enumerator kQSPI_AHB0BufferNotEmpty
AHB buffer 0 not empty

enumerator kQSPI_AHBTransactionPending
AHB access transaction pending

enumerator kQSPI_AHBCommandPriorityGranted
AHB command priority granted

enumerator kQSPI_AHBAccess
AHB access

enumerator kQSPI_IPAccess
IP access

enumerator kQSPI_Busy
Module busy

enumerator kQSPI_StateAll
All flags

enum _qspi_interrupt_enable
QSPI interrupt enable.

Values:

enumerator kQSPI_DataLearningFailInterruptEnable
Data learning pattern failure interrupt enable

enumerator kQSPI_TxBufferFillInterruptEnable
Tx buffer fill interrupt enable

enumerator kQSPI_TxBufferUnderrunInterruptEnable
Tx buffer underrun interrupt enable

enumerator kQSPI_IllegalInstructionInterruptEnable
Illegal instruction error interrupt enable

enumerator kQSPI_RxBufferOverflowInterruptEnable
Rx buffer overflow interrupt enable

enumerator kQSPI_RxBufferDrainInterruptEnable
Rx buffer drain interrupt enable

enumerator kQSPI_AHBSequenceErrorInterruptEnable
AHB sequence error interrupt enable

enumerator kQSPI_AHBBufferOverflowInterruptEnable
AHB buffer overflow interrupt enable

enumerator kQSPI_IPCommandUsageErrorInterruptEnable
IP command usage error interrupt enable

enumerator kQSPI_IPCommandTriggerDuringAHBAccessInterruptEnable
IP command trigger during AHB access error

enumerator kQSPI_IPCommandTriggerDuringIPAccessInterruptEnable
IP command trigger cannot be executed

enumerator kQSPI_IPCommandTriggerDuringAHBGrantInterruptEnable
IP command trigger during AHB grant error

enumerator kQSPI_IPCommandTransactionFinishedInterruptEnable
IP command transaction finished interrupt enable

enumerator kQSPI_AllInterruptEnable
All error interrupt enable

enum _qspi_dma_enable
QSPI DMA request flag.

Values:

enumerator kQSPI_RxBufferDrainDMAEnable
Rx buffer drain DMA

enumerator kQSPI_AllDDMAEnable

enum _qspi_dqs_phrase_shift
Phrase shift number for DQS mode.

Values:

enumerator kQSPI_DQSNoPhraseShift
No phase shift

enumerator kQSPI_DQSPhraseShift45Degree
Select 45 degree phase shift

enumerator kQSPI_DQSPhraseShift90Degree
Select 90 degree phase shift

enumerator kQSPI_DQSPhraseShift135Degree
Select 135 degree phase shift

enum _qspi_dqs_read_sample_clock
Qspi read sampling option.

Values:

enumerator kQSPI_ReadSampleClkInternalLoopback
Read sample clock adopts internal loopback mode.

enumerator kQSPI_ReadSampleClkLoopbackFromDqsPad
Dummy Read strobe generated by QSPI Controller and loopback from DQS pad.

enumerator kQSPI_ReadSampleClkExternalInputFromDqsPad
Flash provided Read strobe and input from DQS pad.

typedef enum *_qspi_read_area* qspi_read_area_t
QSPI read data area, from IP FIFO or AHB buffer.

typedef enum *_qspi_command_seq* qspi_command_seq_t
QSPI command sequence type.

typedef enum *_qspi_fifo* qspi_fifo_t
QSPI buffer type.

typedef enum *_qspi_dqs_phrase_shift* qspi_dqs_phrase_shift_t
Phrase shift number for DQS mode.

typedef enum *_qspi_dqs_read_sample_clock* qspi_dqs_read_sample_clock_t
Qspi read sampling option.

typedef struct *QspiDQSConfig* qspi_dqs_config_t
DQS configure features.

typedef struct *QspiFlashTiming* qspi_flash_timing_t
Flash timing configuration.

typedef struct *QspiConfig* qspi_config_t
QSPI configuration structure.

typedef struct *_qspi_flash_config* qspi_flash_config_t
External flash configuration items.

typedef struct *_qspi_transfer* qspi_transfer_t
Transfer structure for QSPI.

typedef struct *_ip_command_config* ip_command_config_t
16-bit access reg for IPCR register

typedef struct *_qspi_delay_chain_config* qspi_delay_chain_config_t
Slave delay chain configuration items.

QSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
Macro functions for LUT table.

QSPI_CMD
Macro for QSPI LUT command.

QSPI_ADDR

QSPI_DUMMY

QSPI_MODE

QSPI_MODE2

QSPI_MODE4

QSPI_READ

QSPI_WRITE

QSPI_JMP_ON_CS

QSPI_ADDR_DDR

QSPI_MODE_DDR

QSPI_MODE2_DDR

QSPI_MODE4_DDR

QSPI_READ_DDR

QSPI_WRITE_DDR

QSPI_DATA_LEARN

QSPI_CMD_DDR

QSPI_CADDR

QSPI_CADDR_DDR

QSPI_STOP

QSPI_PAD_1
Macro for QSPI PAD.

QSPI_PAD_2

QSPI_PAD_4

QSPI_PAD_8

struct QspiDQSConfig
#include <fsl_qspi.h> DQS configure features.

Public Members

uint32_t portADelayTapNum
Delay chain tap number selection for QSPI port A DQS

qspi_dqs_phrase_shift_t shift
Phase shift for internal DQS generation

qspi_dqs_read_sample_clock_t rxSampleClock
Read sample clock for Dqs.

bool enableDQSClkInverse
Enable inverse clock for internal DQS generation

struct QspiFlashTiming
#include <fsl_qspi.h> Flash timing configuration.

Public Members

uint32_t dataHoldTime
Serial flash data in hold time

uint32_t CSHoldTime
Serial flash CS hold time in terms of serial flash clock cycles

uint32_t CSSetupTime
Serial flash CS setup time in terms of serial flash clock cycles

struct QspiConfig
#include <fsl_qspi.h> QSPI configuration structure.

Public Members

uint8_t txWatermark
QSPI transmit watermark value

uint8_t rxWatermark
QSPI receive watermark value.

uint32_t AHBbufferSize[1]
AHB buffer size.

uint8_t AHBbufferMaster[1]
AHB buffer master.

bool enableAHBbuffer3AllMaster
Is AHB buffer3 for all master.

qspi_read_area_t area
Which area Rx data readout

bool enableQspi
Enable QSPI after initialization

struct __qspi__flash__config
#include <fsl_qspi.h> External flash configuration items.

Public Members

uint32_t flashA1Size
Flash A1 size

uint32_t flashA2Size
Flash A2 size

uint32_t flashB1Size
Flash B1 size

uint32_t flashB2Size
Flash B2 size

uint32_t lookuptable[1]
Flash command in LUT

uint32_t CSHoldTime
CS line hold time

uint32_t CSSetupTime
CS line setup time

uint32_t cloumnspace
Column space size

uint32_t dataLearnValue
Data Learn value if enable data learn

bool enableWordAddress
If enable word address.

struct __qspi__transfer
#include <fsl_qspi.h> Transfer structure for QSPI.

Public Members

uint32_t *data
Pointer to data to transmit

size_t dataSize
Bytes to be transmit

struct __ip__command__config
#include <fsl_qspi.h> 16-bit access reg for IPCR register

struct _qspi_delay_chain_config

#include <fsl_qspi.h> Slave delay chain configuration items.

Public Members

bool highFreqDelay

Selects delay chain for low/high frequency of operation.

union IPCR_REG

Public Members

___IO uint32_t IPCR

IP Configuration Register

struct *_ip_command_config* BITFIELD

struct BITFIELD

Public Members

___IO uint16_t IDATZ

16-bit access for IDATZ field in IPCR register

___IO uint8_t RESERVED_0

8-bit access for RESERVED_0 field in IPCR register

___IO uint8_t SEQID

8-bit access for SEQID field in IPCR register

2.53 Quad Serial Peripheral Interface EDMA Driver

void QSPI_TransferTxCreateHandleEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the QSPI handle for send which is used in transactional functions and set the callback.

Parameters

- base – QSPI peripheral base address
- handle – Pointer to qspi_edma_handle_t structure
- callback – QSPI callback, NULL means no callback.
- userData – User callback function data.
- dmaHandle – User requested eDMA handle for eDMA transfer

void QSPI_TransferRxCreateHandleEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the QSPI handle for receive which is used in transactional functions and set the callback.

Parameters

- base – QSPI peripheral base address
- handle – Pointer to qspi_edma_handle_t structure
- callback – QSPI callback, NULL means no callback.
- userData – User callback function data.
- dmaHandle – User requested eDMA handle for eDMA transfer

status_t QSPI_TransferSendEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_transfer_t* *xfer)

Transfers QSPI data using an eDMA non-blocking method.

This function writes data to the QSPI transmit FIFO. This function is non-blocking.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure
- xfer – QSPI transfer structure.

status_t QSPI_TransferReceiveEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *qspi_transfer_t* *xfer)

Receives data using an eDMA non-blocking method.

This function receive data from the QSPI receive buffer/FIFO. This function is non-blocking. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure
- xfer – QSPI transfer structure.

void QSPI_TransferAbortSendEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – QSPI peripheral base address.
- handle – Pointer to qspi_edma_handle_t structure

void QSPI_TransferAbortReceiveEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle)

Aborts the receive data using eDMA.

This function abort receive data which using eDMA.

Parameters

- base – QSPI peripheral base address.
- handle – Pointer to qspi_edma_handle_t structure

status_t QSPI_TransferGetSendCountEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, size_t *count)

Gets the transferred counts of send.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure.

- 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 QSPI_TransferGetReceiveCountEDMA(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, size_t *count)

Gets the status of the receive transfer.

Parameters

- base – Pointer to QuadSPI Type.
- handle – Pointer to qspi_edma_handle_t structure
- 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_QSPI_EDMA_DRIVER_VERSION

QSPI EDMA driver version.

typedef struct *_qspi_edma_handle* qspi_edma_handle_t

typedef void (*qspi_edma_callback_t)(QuadSPI_Type *base, *qspi_edma_handle_t* *handle, *status_t* status, void *userData)

QSPI eDMA transfer callback function for finish and error.

struct _qspi_edma_handle

#include <fsl_qspi_edma.h> QSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

eDMA handler for QSPI send

size_t transferSize

Bytes need to 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 QSPI eDMA transfer

qspi_edma_callback_t callback

Callback for users while transfer finish or error occurred

void *userData

User callback parameter

2.54 RTC: Real Time Clock

FSL_RTC_DRIVER_VERSION

Version 2.0.0

MINIMUM_RTCVAL

Minimum RTC compare value. Program RTCVAL register with the value greater than 4 if RTCF related \ functionality is required

MINIMUM_APIVAL

Minimum API compare value. Program APIVAL register with the value greater than 4 if APIF related \ functionality is required

enum _rtc_clock_divide

List of RTC clock divide.

Values:

enumerator kRTC_ClockDivide1

Clock divide 1

enumerator kRTC_ClockDivide32

Clock divide 32

enumerator kRTC_ClockDivide512

Clock divide 512

enumerator kRTC_ClockDivide16384

Clock divide 16384

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_APIInterruptEnable

API interrupt enable, bit 14

enumerator kRTC_AutonomousPeriodicInterruptEnable

Autonomous periodic interrupt enable, bit 15

enumerator kRTC_CounterRollOverInterruptEnable

Counter roll over interrupt Enable, bit 28

enumerator kRTC_RTCInterruptEnable

RTC interrupt enable, bit 30

enumerator kRTC_AllInterruptEnable

All interrupt enable

enum _RTC_status_flags

List of RTC Interrupt flags.

Values:

enumerator kRTC_CounterRollOverInterruptFlag

Counter roll over interrupt flag, bit 10.

enumerator kRTC_APIInterruptFlag

API interrupt flag, bit 13.

enumerator kRTC_InvalidAPIFlag

Invalid APIVAL write flag, bit 17.

enumerator kRTC_InvalidRTCFlag
Invalid RTCVAL write flag, bit 18.

enumerator kRTC_RTCInterruptFlag
RTC interrupt flag, bit 29.

enumerator kRTC_AllInterruptFlags
All interrupt flags

enumerator kRTC_AllStatusFlags
All status flags

enum rtc_callback_type_t
Callback type when registering for a callback. It tells the call back is RTC, API or counter roll over call back.

Values:

enumerator kRTC_APICallback
API interrupt callback

enumerator kRTC_CounterRollOverCallback
Counter roll over interrupt callback

enumerator kRTC_RTCCallback
RTC interrupt callback

typedef enum *_rtc_clock_divide* rtc_clock_divide_t
List of RTC clock divide.

typedef enum *_rtc_interrupt_enable* rtc_interrupt_enable_t
List of RTC interrupts.

typedef enum *_RTC_status_flags* rtc_status_flags_t
List of RTC Interrupt flags.

typedef struct *_rtc_config* rtc_config_t
RTC config structure.

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

typedef void (*rtc_callback_t)(*rtc_callback_type_t* type)
RTC callback function.

struct __rtc_config
#include <fsl_rtc.h> RTC config structure.

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

2.55 SAI: Serial Audio Interface

2.56 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_SetMasterClockConfig(I2S_Type *base, *sai_master_clock_t* *config)

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

void SAI_TxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.

- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly    = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

void SAI_TxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer

- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer

- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

void SAI_TransferTxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SAI_TransferRxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

void SAI_TransferTxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – tranmitter configurations.

void SAI_TransferRxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.

- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.10

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy

SAI Tx is busy.

enumerator kStatus_SAI_RxBusy

SAI Rx is busy.

enumerator kStatus_SAI_TxError

SAI Tx FIFO error.

enumerator kStatus_SAI_RxError

SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull

SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle

SAI Tx is idle

enumerator kStatus_SAI_RxIdle

SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.

Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.

Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.

Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.

Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.

Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.

Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx
Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx
Synchronous with another SAI receiver

enum _sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1

Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable

FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable

FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable

FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable

FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable

FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing
The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate
Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz
Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz
Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz
Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz
Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz
Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz
Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz
Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz
Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz
Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz
Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width
Audio word width.

Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_data_pin_state
sai data pin state definition

Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine
sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled
sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead
sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite
sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite
sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead
sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite
sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type
sai transceiver type

Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
: sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
: sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
: sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
: sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
: sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
: master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
: sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
: sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
: sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
: sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
: sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
: SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)
: SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK
: Used to control whether SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

  If this macro is set to 0 then SAI_RxSetFifoConfig()/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG
: Disable implicit channel data configuration within SAI_TxSetConfig()/SAI_RxSetConfig().

  Use this macro to control whether SAI_RxSetConfig()/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, SAI_TxSetConfig()/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE
: SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE
: sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol
: Audio bus protocol in SAI

sai_sync_mode_t syncMode
: SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable
: Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource
: Bit Clock source

sai_master_slave_t masterSlave
: Master or slave

struct _sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz
: Sample rate of audio data

uint32_t bitWidth
: Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo
: Mono or stereo

uint32_t masterClockHz
: Master clock frequency in Hz

uint8_t watermark
: Watermark value

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct _sai_master_clock

#include <fsl_sai.h> master clock configurations

Public Members

bool mclkOutputEnable
: master clock output enable

uint32_t mclkHz
: target mclk frequency

uint32_t mclkSourceClkHz
: mclk source frequency

struct _sai_fifo
: *#include <fsl_sai.h>* sai fifo configurations

Public Members

bool fifoContinueOneError
: fifo continues when error occur

sai_fifo_combine_t fifoCombine
: fifo combine mode

sai_fifo_packing_t fifoPacking
: fifo packing mode

uint8_t fifoWatermark
: fifo watermark

struct _sai_bit_clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkSrcSwap
: bit clock source swap

bool bclkInputDelay
: bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
: bit clock polarity

sai_bclk_source_t bclkSource
: bit Clock source

struct _sai_frame_sync
: *#include <fsl_sai.h>* sai frame sync configurations

Public Members

uint8_t frameSyncWidth
: frame sync width in number of bit clocks

bool frameSyncEarly
: TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity
frame sync polarity

struct __sai__serial__data
#include <fsl_sai.h> sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
configure the number of bits in the first word in each frame

uint8_t dataWordNLength
configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
used to record the data length for dma transfer

uint8_t dataFirstBitShifted
Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
configure the number of words in each frame

uint32_t dataMaskedWord
configure whether the transmit word is masked

struct __sai__transceiver
#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct __sai__transfer
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct __sai__handle
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
Transfer status

sai_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32 bits

uint8_t channel
Transfer start channel

uint8_t channelMask
enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

uint8_t watermark
Watermark value

2.57 SAI EDMA Driver

void SAI_TransferTxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle)

Initializes the SAI eDMA handle.

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- txDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferRxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *rxDmaHandle)

Initializes the SAI Rx eDMA handle.

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- rxDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferSetInterleaveType(*sai_edma_handle_t* *handle, *sai_edma_interleave_t* interleaveType)

Initializes the SAI interleave type.

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI_EDMAInterleavePerChannelSample always

Parameters

- handle – SAI eDMA handle pointer.
- interleaveType – Multi channel interleave type.

void SAI_TransferTxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Tx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnWrite to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

void SAI_TransferRxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Rx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;
SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

status_t SAI_TransferSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers

b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SAI is busy sending data.

status_t SAI_TransferReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using eDMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers

b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SAI is busy receiving data.

status_t SAI_TransferSendLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.

status_t SAI_TransferReceiveLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.

void SAI_TransferTerminateSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI transfer using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

status_t SAI_TransferGetSendCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count sent by SAI.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- count – Bytes count sent by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

status_t SAI_TransferGetReceiveCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count received by SAI.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

- count – Bytes count received by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

uint32_t SAI_TransferGetValidTransferSlotsEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Gets valid transfer slot.

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

Return values

valid – slot count that application submit.

FSL_SAI_EDMA_DRIVER_VERSION

Version 2.7.3

enum _sai_edma_interleave

sai interleave type

Values:

enumerator kSAI_EDMAInterleavePerChannelSample

enumerator kSAI_EDMAInterleavePerChannelBlock

typedef struct *sai_edma_handle* sai_edma_handle_t

typedef void (*sai_edma_callback_t)(I2S_Type *base, *sai_edma_handle_t* *handle, *status_t* status, void *userData)

SAI eDMA transfer callback function for finish and error.

typedef enum *_sai_edma_interleave* sai_edma_interleave_t

sai interleave type

MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferReceiveEDMA

MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferSendEDMA

struct sai_edma_handle

#include <fsl_sai_edma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for SAI send

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t bytesPerFrame
: Bytes in a frame

uint8_t channelMask
: Enabled channel mask value, reference _sai_channel_mask

uint8_t channelNums
: total enabled channel nums

uint8_t channel
: Which data channel

uint8_t count
: The transfer data count in a DMA request

uint32_t state
: Internal state for SAI eDMA transfer

sai_edma_callback_t callback
: Callback for users while transfer finish or error occurs

void *userData
: User callback parameter

uint8_t tcd[((4U) + 1U) * sizeof(*edma_tcd_t*)]
: TCD pool for eDMA transfer.

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer.

size_t transferSize[(4U)]
: Data bytes need to transfer

sai_edma_interleave_t interleaveType
: Transfer interleave type

volatile uint8_t queueUser
: Index for user to queue transfer.

volatile uint8_t queueDriver
: Index for driver to get the transfer data and size

2.58 SAR_ADC: SAR_ADC Module

void ADC_GetDefaultConfig(*adc_config_t* *config)
: This function is used to get available predefined configurations for the ADC initialization.

  Parameters

  - config – Pointer to the ADC configuration structure, please refer to adc_config_t for details.

void ADC_Init(ADC_Type *base, const *adc_config_t* *config)
: This function is used to initialize the ADC.

  Parameters

  - base – ADC peripheral base address.
  - config – Pointer to the ADC configuration structure, please refer to adc_config_t for details.

void ADC_Deinit(ADC_Type *base)

This function is used to de-initialize the ADC.

Parameters

- base – ADC peripheral base address.

static inline void ADC_SetPowerDownMode(ADC_Type *base, bool enable)

This function is used to enter or exit power-down mode.

After the release of the reset, the ADC analog module will be kept in power-down mode by default. The power-down mode can be set anytime. However, ADC can enter the power-down mode successfully only after completion of an ongoing conversion (if there is one). In scan mode, the ongoing operation should be aborted manually before or after switching mode. If the power-down mode is entered by setting MCR[PWDN], the process running in the previous mode must be restarted manually (by setting the appropriate START bit in the MCR register) after exiting power-down mode.

Note: After setting the ADC mode, it is recommended to use the function ADC_GetAdcState to query whether the ADC has correctly entered the mode.

Parameters

- base – ADC peripheral base address.
- enable – Indicates whether to enter or exit power-down mode.
  - **true** Request to enter power-down mode.
  - **false** When ADC status is in power-down mode (MSR[ADCSTATUS] = 001b), start ADC transition to IDLE mode.

static inline void ADC_SetOperatingClock(ADC_Type *base, *adc_clock_frequency_t* clockSelect)

This function is used to select the ADC operating clock.

Note: Needs to enter power-down mode before changing the ADC internal operating clock.

Parameters

- base – ADC peripheral base address.
- clockSelect – ADC clock frequency selection, please refer to adc_clock_frequency_t for details.

static inline void ADC_SetAdcSpeedMode(ADC_Type *base, *adc_speed_mode_t* speedMode)

This function is used to set the ADC speed mode.

Parameters

- base – ADC peripheral base address.
- speedMode – ADC speed mode selection, please refer to adc_speed_mode_t for details.

static inline *adc_state_t* ADC_GetAdcState(ADC_Type *base)

This function is used to get the ADC state.

Parameters

- base – ADC peripheral base address.

Returns

ADC state, for possible states, please refer to adc_state_t for details.

static inline bool ADC_CheckAutoClockOffEnabled(ADC_Type *base)

This function is used to check whether the ADC auto clock-off feature has been enabled or not.

Parameters

- base – ADC peripheral base address.

Returns

ADC auto clock-off feature status.

- **true** Auto clock-off feature has been enabled.
- **false** Auto clock-off feature has not been enabled.

static inline uint8_t ADC_GetCurrentConvertedChannelId(ADC_Type *base)

This function is used to get the ID of the channel that is currently being converted.

Parameters

- base – ADC peripheral base address.

Returns

ADC channel ID that is currently being converted.

static inline bool ADC_CheckSelfTestConvInProcess(ADC_Type *base)

This function is used to check whether the self-test conversion is in process or not.

Parameters

- base – ADC peripheral base address.

Returns

Self-test conversion status.

- **true** Self-test conversion is in process.
- **false** Self-test conversion is not in process.

static inline bool ADC_CheckBctuConvStatus(ADC_Type *base)

This function is used to check whether the BCTU conversion was started.

Parameters

- base – ADC peripheral base address.

Returns

BCTU conversion status.

- **true** Ongoing conversion was triggered by BCTU.
- **false** Conversion was not triggered by BCTU.

static inline bool ADC_CheckInjectConvInProcess(ADC_Type *base)

This function is used to check whether the inject conversion is in process or not.

Parameters

- base – ADC peripheral base address.

Returns

Inject conversion status.

- **true** Inject conversion is in process.
- **false** Inject conversion is not in process.

static inline bool ADC_CheckInjectConvAborted(ADC_Type *base)

This function is used to check whether the inject conversion has been aborted or not.

Parameters

- base – ADC peripheral base address.

Returns
Inject conversion abort status.

- **true** Injected conversion has been aborted.
- **false** Injected conversion has not been aborted.

static inline bool ADC_CheckNormalConvInProcess(ADC_Type *base)

This function is used to check whether the normal conversion is in process or not.

Parameters

- base – ADC peripheral base address.

Returns
Normal conversion status.

- **true** Normal conversion is in process.
- **false** Normal conversion is not in process.

static inline bool ADC_CheckCalibrationBusy(ADC_Type *base)

This function is used to check whether the ADC is executing calibration or ready for use.

Parameters

- base – ADC peripheral base address.

Returns
Calibration process status.

- **true** ADC is busy in a calibration process.
- **false** ADC is ready for use.

static inline bool ADC_CheckCalibrationFailed(ADC_Type *base)

This function is used to check whether the calibration has failed or passed.

Note: When the user clears the calibration failed status and then reads the status, it will display the calibration passed. At this time, the calibration may not be successful. The user must read the MSR[CALBUSY] bit by function ADC_CheckCalibrationBusy to perform a double check.

Returns
Normal conversion status.

- **true** Calibration failed.
- **false** Calibration passed (must be checked with CALBUSY = 0b).

static inline void ADC_ClearCalibrationFailedFlag(ADC_Type *base)

This function is used to clear the flag of calibration.

Parameters

- base – ADC peripheral base address.

static inline bool ADC_CheckCalibrationSuccessful(ADC_Type *base)

This function is used to check whether the calibration is successful or not.

Parameters

- base – ADC peripheral base address.

Returns
Normal conversion status.

- **true** Calibrated or calibration successful.
- **false** Uncalibrated or calibration unsuccessful.

void ADC_SetConvChainConfig(ADC_Type *base, const *adc_chain_config_t* *config)

This function is used to configure the chain.

Parameters

- base – ADC peripheral base address.
- config – Pointer to the chain configuration structure, please refer to adc_chain_config_t for details.

static inline void ADC_EnableSpecificChannelNormalConv(ADC_Type *base, uint8_t channelIndex)

This function is used to enable the specific ADC channel to execute normal conversion.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the normal conversion.

static inline void ADC_DisableSpecificChannelNormalConv(ADC_Type *base, uint8_t channelIndex)

This function is used to disable the specific ADC channel to execute the normal conversion.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the normal conversion.

static inline void ADC_EnableSpecificChannelInjectConv(ADC_Type *base, uint8_t channelIndex)

This function is used to enable the specific ADC channel to execute the inject conversion.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the inject conversion.

static inline void ADC_DisableSpecificChannelInjectConv(ADC_Type *base, uint8_t channelIndex)

This function is used to disable the specific ADC channel to execute the inject conversion.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the inject conversion.

static inline void ADC_SetConvMode(ADC_Type *base, *adc_conv_mode_t* convMode)

This function is used to set the ADC conversion mode.

Note: Before setting the conversion mode, users need to check whether the ADC is in the idle status through the function ADC_GetAdcState.

Parameters

- base – ADC peripheral base address.
- convMode – ADC conversion mode, please refer to adc_conv_mode_t for details.

static inline void ADC_StartConvChain(ADC_Type *base, *adc_conv_mode_t* convMode)

This function is used to start the ADC conversion chain to execute the conversion.

Note: Normal conversion supports two conversion modes, one is one-shot conversion mode, and the other is scan conversion mode. Normal conversion should usually be used to convert analog samples most of the time in an application, unless there is a special need. Inject conversion has a higher priority than normal conversion and it runs in one-shot mode only, it can be started in IDLE condition or when normal conversion is in process.

Parameters

- base – ADC peripheral base address.
- convMode – Pointer to the ADC conversion chain, please refer to adc_conv_mode_t for details.

static inline void ADC_StopConvChain(ADC_Type *base)

This function is used to stop scan in normal conversion scan operation mode.

Note: In scan operation mode, the MCR[NSTART] field remains high after setting. Clearing this field in scan operation mode causes the current chain conversion to finish, then stop the scan.

Parameters

- base – ADC peripheral base address.

static inline void ADC_AbortCurrentConvChain(ADC_Type *base)

This function is used to abort the conversion chain.

Abort the current chain of conversions by setting MCR[ABORTCHAIN]. In that case, the behavior of the ADC depends on MCR[MODE] (one-shot/scan conversion modes). In one-shot mode, MSR[NSTART] is automatically reset together with MCR[ABORTCHAIN], an end-of-chain interrupt is not generated in the case of an abort chain. In scan mode, a new chain is started. The end-of-conversion interrupt of the current aborted conversion is not generated but an end-of-chain interrupt is generated.

Note: Setting this field in an IDLE state (for example, no normal/inject conversion is in process) has no effect. In this case, the MCR[ABORTCHAIN] field is reset immediately.

Parameters

- base – ADC peripheral base address.

static inline void ADC_AbortCurrentConv(ADC_Type *base)

This function is used to abort the conversion channel.

Abort the current conversion and immediately start the conversion of the next channel of the chain. In the case of an abort operation, MSR[NSTART/JSTART] remains set if not in the last channel of the chain, and MCR[ABORT] is reset as soon as the channel is aborted. The end-of-conversion interrupt corresponds to the aborted channel is not generated. This behavior is true for normal or inject conversion modes. If the last channel of a chain is aborted, and the end-of-chain is reported, then an end-of-chain interrupt will be generated.

Note: This conversion can not abort while the self-test channel conversion is in process.

Parameters

- base – ADC peripheral base address.

static inline void ADC_EnableConvInt(ADC_Type *base, uint32_t mask)

This function is used to enable the ADC end-of-conversion and end-of-chain interrupts.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to enable the ADC end-of-conversion and end-of-chain interrupts, please refer to _adc_conv_int_enable for details.

static inline void ADC_DisableConvInt(ADC_Type *base, uint32_t mask)

This function is used to disable the ADC end-of-conversion and end-of-chain interrupts.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to disable the ADC end-of-conversion and end-of-chain interrupts, please refer to _adc_conv_int_enable for details.

static inline uint32_t ADC_GetConvIntStatus(ADC_Type *base)

This function is used to get the ADC end-of-conversion and end-of-chain interrupts status.

Parameters

- base – ADC peripheral base address.

Returns

ADC end-of-conversion and end-of-chain interrupts status mask.

static inline void ADC_ClearConvIntStatus(ADC_Type *base, uint32_t mask)

This function is used to clear the ADC end-of-conversion and end-of-chain interrupts status.

Parameters

- base – ADC peripheral base address.
- mask – Mask value for flags to be cleared, please refer to _adc_conv_int_flag for details.

static inline void ADC_EnableSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)

This function is used to enable the specific ADC channel end-of-conversion interrupt.

Note: This function can only turn on the interrupt of a specific channel, if the interrupt occurs, the user also needs to turn on the global end-of-conversion interrupt by using function ADC_EnableConvInt

---

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the end-of-conversion interrupt.

static inline void ADC_DisableSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)

This function is used to disable the specific ADC channel end-of-conversion interrupt.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the end-of-conversion interrupt.

static inline bool ADC_CheckSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)

This function is used to check whether the specific conversion channel's end-of-conversion interrupt has been occured.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to check the end-of-conversion interrupt status.

Returns

Channel end-of-conversion interrupt status flag of the specific channel.

- **true** Channel end-of-conversion interrupt has been occured.
- **false** Channel end-of-conversion interrupt has not been occured.

static inline void ADC_ClearSpecificConvChannelInt(ADC_Type *base, uint8_t channelIndex)

This function is used to clear specific channel end-of-conversion interrupt flag.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to clear the end-of-conversion interrupt flag.

static inline void ADC_EnableDmaTransfer(ADC_Type *base)

This function is used to enable the DMA transfer function.

Note: This function is a master switch used to control whether the data converted by the ADC is transmitted through DMA. If the user configures DMA to transmit the conversion data of the channel during the chain channel configuration process, then the main switch of the DMA transmission must be turned on, otherwise, data transmission cannot be performed.

Parameters

- base – ADC peripheral base address.

static inline void ADC_DisableDmaTransfer(ADC_Type *base)

This function is used to disable the DMA transfer function.

Parameters

- base – ADC peripheral base address.

static inline void ADC_EnableSpecificConvChannelDmaTransfer(ADC_Type *base, uint8_t channelIndex)

This function is used to enable the specific ADC conversion channel's DMA transfer function.

Note: This function can only turn on the DMA transfer of a specific channel, before using the DMA transfer, the user needs to turn on the global DMA transfer by using the function ADC_EnableDmaTransfer.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the DMA transfer function.

static inline void ADC_DisableSpecificConvChannelDmaTransfer(ADC_Type *base, uint8_t channelIndex)

This function is used to disable the specific ADC conversion channel's DMA transfer function.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the DMA transfer function.

static inline void ADC_EnableSpecificConvChannelPresample(ADC_Type *base, uint8_t channelIndex)

This function is used to enable the specific ADC conversion channel's pre-sample function.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to enable the pre-sample function.

static inline void ADC_DisableSpecificConvChannelPresample(ADC_Type *base, uint8_t channelIndex)

This function is used to disable the specific ADC conversion channel's pre-sample function.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to disable the pre-sample function.

void ADC_SetAnalogWdgConfig(ADC_Type *base, const *adc_wdg_config_t* *config)

This function is used to configure the analog watchdog.

The analog watchdogs are used to monitor the conversion result to see if it is within defined limits, specified by a higher and a lower threshold value. After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value is outside the threshold values, then a corresponding threshold violation interrupt is generated.

Parameters

- base – ADC peripheral base address.
- config – Pointer to the analog watchdog configuration structure, please refer to adc_wdg_config_t for details.

static inline void ADC_EnableSpecificConvChannelAnalogWdg(ADC_Type *base, uint8_t channelIndex)

This function is used to enable the specific ADC conversion channel's analog watchdog function.

Parameters

- base – ADC peripheral base address.
- channelIndex – Conversion channel index to enable the analog watchdog function.

static inline void ADC_DisableSpecificConvChannelAnalogWdg(ADC_Type *base, uint8_t channelIndex)

This function is used to disable the specific ADC conversion channel's analog watchdog function.

Parameters

- base – ADC peripheral base address.

- channelIndex – Conversion channel index to disable the analog watchdog function.

static inline bool ADC_CheckSpecificConvChannelOutofRange(ADC_Type *base, uint8_t channelIndex)

This function is used to check whether the specific conversion channel's converted data is out of range.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to check the converted data out of range status.

Returns

Converted data out of range status of the specific conversion channel.

- **true** Channel converted data is out of range.
- **false** Channel converted data is in range.

static inline void ADC_ClearSpecificConvChannelOutofRange(ADC_Type *base, uint8_t channelIndex)

This function is used to clear the specific conversion channel's converted data out-of-range flag.

Parameters

- base – ADC peripheral base address.
- channelIndex – Channel index to clear the converted data out of range flag.

static inline void ADC_EnableWdgThresholdInt(ADC_Type *base, uint32_t mask)

This function is used to enable the analog watchdog threshold low or/and high interrupts.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to enable the analog watchdog threshold low or/and high interrupts, please refer to _adc_wdg_threshold_int_enable for details.

static inline void ADC_DisableWdgThresholdInt(ADC_Type *base, uint32_t mask)

This function is used to disable the analog watchdog threshold low or/and high interrupts.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to disable the analog watchdog threshold low or/and high interrupts, please refer to _adc_wdg_threshold_int_enable for details.

static inline uint32_t ADC_GetWdgThresholdIntStatus(ADC_Type *base)

This function is used to get the analog watchdog threshold interrupts status.

Parameters

- base – ADC peripheral base address.

Returns

Analog watchdog threshold interrupts status mask.

static inline void ADC_ClearWdgThresholdIntStatus(ADC_Type *base, uint32_t mask)

This function is used to clear the analog watchdog threshold low or/and high interrupts status.

Parameters

- base – ADC peripheral base address.

- mask – Mask value for flags to be cleared, please refer to _adc_wdg_threshold_int_flag for details.

bool ADC_DoCalibration(ADC_Type *base, const *adc_calibration_config_t* *config)

This function is used to do the calibration.

The calibration is used to reduce or eliminate the various errors. In the calibration process, the calibration values for offset, gain, and capacitor mismatch are obtained. These calibration values (except gain calibration) are used in a result post-processing step to reduce or eliminate the various errors contribution effects. The gain calibration is used during the sample phase to define the additional charge to be loaded in order to compensate for the gain failure. Calibration must be performed after every power-up reset and whenever required in runtime operation. It is also recommended to run calibration if the operating conditions (particularly VrefH) change. Never apply functional reset during the calibration process. If applied, calibration must be rerun after exiting a reset condition; otherwise, the calibration-generated values and conversion results may be unspecified.

Note: This function executes a calibration sequence, it is recommended to run this sequence before using the ADC converter. The maximum clock frequency for the calibration is 40 MHz. Before calling this function, the user needs to ensure that the input clock is within 40MHz. The results of individual steps are also updated in the CALSTAT register (CALSTAT[STAT_n]). The result of the last failed step is dynamically updated in the same register.

Parameters

- base – ADC peripheral base address.
- config – Pointer to the calibration configuration structure, please refer to adc_calibration_config_t for details.

Returns

Status whether calibration is running passed or failed.

- **true** Calibration successful.
- **false** Calibration unsuccessful.

void ADC_SetUserOffsetAndGainConfig(ADC_Type *base, const *adc_user_offset_gain_config_t* *config)

This function is used to configure the user gain and offset.

Parameters

- base – ADC peripheral base address.
- config – Pointer to the user offset and gain configuration structure, please refer to adc_user_offset_gain_config_t for details.

void ADC_SetSelfTestConfig(ADC_Type *base, const *adc_self_test_config_t* *config)

This function is used to configure the ADC self-test.

The self-test is used to check at regular intervals whether ADC is operating correctly. When self-test is enabled, ADC automatically checks its components and flags any errors it finds. The test can be enabled to check the supply voltage (VDD), reference voltage (VrefH), and calibrated values.

Note: Before calling this function, please ensure the functional conversion is one-shot conversion mode normal conversion type and the operating clock are equal to bus frequency. ADC self-test should be run with MCR[ADCLKSE] bit set to 1. Self-test with ADCLKSE bit set to 0 can give erroneous results.

Parameters

- base – ADC peripheral base address.
- config – Pointer to the self-test configuration structure, please refer to adc_self_test_config_t for details.

void ADC_SetSelfTestWdgConfig(ADC_Type *base, const *adc_self_test_wdg_config_t* *config)

This function is used to configure the ADC self-test watchdog.

Parameters

- base – ADC peripheral base address.
- config – Pointer to the self-test watchdog configuration structure, please refer to adc_self_test_wdg_config_t for details.

void ADC_GetCalibrationLastFailedTestResult(ADC_Type *base, int16_t *result)

This function is used to get the test result for the last failed test.

Parameters

- base – ADC peripheral base address.
- result – Points to a 16-bit signed variable, and it is used to store the test result for the last failing test.

static inline uint16_t ADC_GetCalibrationStepsStatus(ADC_Type *base)

This function is used to get the status of the calibration steps.

Note: The status of calibration steps (step 0 to step 12) is stored in the CALSTAT register, and only the lower 12 bits are available in the returned result.

Parameters

- base – ADC peripheral base address.

Returns

ADC self-test interrupt status mask.

static inline void ADC_EnableSelfTest(ADC_Type *base)

This function is used to enable the ADC self-test.

Decides whether to enable the ADC self-test. The self-test test is enabled by setting STCR2[EN], this field must be set before starting normal conversion and should not be changed while the conversion is in process. This field should only be reset after the end-of-conversion of the last self-test channel has been received.

Note: ADC self-test should be run with MCR[ADCLKSE] bit set to 1. Self-test with ADCLKSE bit set to 0 can give erroneous results. In the case of Inject Conversion mode, test channel conversion is not performed. It is performed only during normal conversions.

Parameters

- base – ADC peripheral base address.

static inline void ADC_DisableSelfTest(ADC_Type *base)

This function is used to disable the ADC self-test.

Parameters

- base – ADC peripheral base address.

static inline void ADC_EnableSelfTestWdgThreshold(ADC_Type *base,
adc_self_test_wdg_threshold_t wdgID)

This function is used to enable the ADC self-test watchdog threshold for algorithm S step 0/1/2 and algorithm C.

The user can pass kADC_SelfTestWdgThresholdForAlgSStep0 as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 0; pass kADC_SelfTestWdgThresholdForAlgSStep1Integer as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 1; pass kADC_SelfTestWdgThresholdForAlgSStep2 as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm S step 2; pass kADC_SelfTestWdgThresholdForAlgCStep0 as parameter 'wdgID' to enable the self-test watchdog threshold function for algorithm C; Other enumerations in adc_self_test_wdg_threshold_t have no use.

Parameters

- base – ADC peripheral base address.
- wdgID – Watchdog threshold index to enable, please refer to adc_self_test_wdg_threshold_t for details.

static inline void ADC_DisableSelfTestWdgThreshold(ADC_Type *base,
adc_self_test_wdg_threshold_t wdgID)

This function is used to disable the ADC self-test watchdog threshold for algorithm S step 0/1/2 and algorithm C.

The user can pass kADC_SelfTestWdgThresholdForAlgSStep0 as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 0; pass kADC_SelfTestWdgThresholdForAlgSStep1Integer as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 1; pass kADC_SelfTestWdgThresholdForAlgSStep2 as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm S step 2; pass kADC_SelfTestWdgThresholdForAlgCStep0 as parameter 'wdgID' to disable the self-test watchdog threshold function for algorithm C; Other enumerations in adc_self_test_wdg_threshold_t have no use.

Parameters

- base – ADC peripheral base address.
- wdgID – Watchdog threshold index to disable, please refer to adc_self_test_wdg_threshold_t for details.

static inline void ADC_EnableSelfTestWdgTimer(ADC_Type *base, *adc_alg_type_t*
wdgTimerType)

This function is used to enable the ADC self-test watchdog timer for algorithm S or/and algorithm C.

The user can pass kADC_SelfTestForAlgS as parameter 'wdgTimerType' to enable the watchdog timer for algorithm S; pass kADC_SelfTestForAlgC as parameter 'wdgTimerType' to enable the watchdog timer for algorithm C; pass kADC_SelfTestForAlgSAndC as parameter 'wdgTimerType' to enable the watchdog timer for algorithm S and C.

Parameters

- base – ADC peripheral base address.
- wdgTimerType – Watchdog timer type to enable, please refer to adc_alg_type_t for details.

static inline void ADC_DisableSelfTestWdgTimer(ADC_Type *base, *adc_alg_type_t*
wdgTimerType)

This function is used to disable the ADC self-test watchdog timer.

The user can pass kADC_SelfTestForAlgS as parameter 'wdgTimerType' to disable the watchdog timer for algorithm S; pass kADC_SelfTestForAlgC as parameter 'wdgTimerType' to disable the watchdog timer for algorithm C; pass kADC_SelfTestForAlgSAndC as parameter 'wdgTimerType' to disable the watchdog timer for algorithm S and C.

Parameters

- base – ADC peripheral base address.
- wdgTimerType – Watchdog timer type to disable, please refer to adc_alg_type_t for details.

static inline void ADC_SetSelfTestWdgTimerVal(ADC_Type *base, *adc_wdg_timer_val_t* wdgTimerVal)

This function is used to set the ADC self-test watchdog timer value.

Parameters

- base – ADC peripheral base address.
- wdgTimerVal – Watchdog timer value, please refer to adc_wdg_timer_val_t for details.

static inline uint16_t ADC_GetSelfTestChannelConvFailedData(ADC_Type *base, *adc_self_test_wdg_threshold_t* type)

This function is used to get the ADC self-test channel converted data when ERR_S0/ERR_S1_INTEGER/ ERR_S1_FRACTION/ERR_S2/ERR_C occurred.

Note: The user can pass kADC_SelfTestWdgThresholdForAlgSStep0 as parameter 'type' to get the converted data when ERR_S0 occurred; pass kADC_SelfTestWdgThresholdForAlgSStep1Integer as parameter 'type' to get the converted data when ERR_S1_INTEGER occurred; pass kADC_SelfTestWdgThresholdForAlgSStep1Fraction as parameter 'type' to get the converted data when ERR_S1_FRACTION occurred; pass kADC_SelfTestWdgThresholdForAlgSStep2 as parameter 'type' to get the converted data when ERR_S2 occurred; pass kADC_SelfTestWdgThresholdForAlgCStep0 as parameter 'type' to get the converted data when ERR_C occurred; Other enumerations in adc_self_test_wdg_threshold_t have no use.

Parameters

- base – ADC peripheral base address.
- type – Watchdog threshold index to get the ADC self-test channel converted data, please refer to adc_self_test_wdg_threshold_t for details.

static inline void ADC_EnableSelfTestInt(ADC_Type *base, uint32_t mask)

This function is used to enable the ADC self-test-related interrupts.

Note: Watchdog timer feature is applicable only for scan operation mode and not for one-shot operation mode.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to enable the ADC self-test related interrupts, please refer to _adc_self_test_int_enable for details.

static inline void ADC_DisableSelfTestInt(ADC_Type *base, uint32_t mask)

This function is used to disable the ADC self-test interrupt.

Parameters

- base – ADC peripheral base address.
- mask – Mask value to disable the ADC self-test related interrupts, please refer to _adc_self_test_int_enable for details.

static inline uint32_t ADC_GetSelfTestIntStatus(ADC_Type *base)

This function is used to get the ADC self-test interrupts status.

Parameters

- base – ADC peripheral base address.

Returns

ADC self-test related interrupts status mask.

static inline void ADC_ClearSelfTestIntStatus(ADC_Type *base, uint32_t mask)

This function is used to clear the ADC self-test interrupts status.

Parameters

- base – ADC peripheral base address.
- mask – Mask value for flags to be cleared, please refer to _adc_self_test_int_flag for details.

bool ADC_GetChannelConvResult(ADC_Type *base, *adc_conv_result_t* *result, uint8_t channelIndex)

This function is used to get the specific ADC channel's conversion result.

CDR[VALID] indicates whether a new conversion is available, this field is automatically reset to 0 when the data is read. CDR[OVERW] Indicates whether the previous conversion data was overwritten without having been read, in which case the overwritten data is lost.

Parameters

- base – SAR ADC peripheral base address.
- result – Pointer to SAR ADC channels conversion result structure, please refer to adc_conv_result_t for details.
- channelIndex – Channel index to get the conversion result.

Returns

Indicates whether the acquisition of the specific channel conversion result is successful or not.

- **true** Obtaining the specific channel conversion result successfully, and the conversion result is stored in the input parameter result.
- **false** Obtaining the specific channel conversion result failed.

bool ADC_GetSelfTestChannelConvData(ADC_Type *base, *adc_self_test_conv_result_t* *result)

This function is used to get the test channel converted data when algorithm S step 0, algorithm S step 2, or algorithm C step executes.

Parameters

- base – ADC peripheral base address.
- result – Pointer to the SAR ADC self-test channel conversion result structure, please refer to adc_self_test_conv_result_t for details.

Returns

Indicates whether the acquisition of the self-test channel conversion result is successful or not.

- **true** Obtaining the self-test channel conversion result successfully, and the conversion result is stored in the input parameter 'result'.
- **false** Obtaining the self-test channel conversion result failed.

bool ADC_GetSelfTestChannelConvDataForAlgSStep1(ADC_Type *base, *adc_self_test_conv_result_t* *result)

This function is used to get the test channel converted data when algorithm S step 1 executes.

Parameters

- base – ADC peripheral base address.
- result – Pointer to the SAR ADC self-test channel conversion result structure, please refer to adc_self_test_conv_result_t for details.

Returns

Indicates whether the acquisition of the self-test channel conversion result is successful or not.

- **true** Obtaining the self-test channel conversion result successfully, and the conversion result is stored in the input parameter 'result'.
- **false** Obtaining the self-test channel conversion result failed.

FSL_SAR_ADC_DRIVER_VERSION

SAR ADC driver version 2.3.0.

enum _adc_conv_int_enable

This enumeration provides the mask for the ADC end-of-conversion and end-of-chain interrupts enabling.

Values:

enumerator kADC_NormalConvChainEndIntEnable

Enable end of normal chain conversion interrupt.

enumerator kADC_NormalConvEndIntEnable

Enable end of normal conversion interrupt.

enumerator kADC_InjectConvChainEndIntEnable

Enable end of inject chain conversion interrupt.

enumerator kADC_InjectConvEndIntEnable

Enable end of inject conversion interrupt.

enumerator kADC_BctuConvEndIntEnable

Enable end of BCTU conversion interrupt.

enum _adc_wdg_threshold_int_enable

This enumeration provides the mask for the ADC analog watchdog threshold interrupts enabling.

Values:

enumerator kADC_wdg0LowThresholdIntEnable

Enable watchdog 0 low threshold interrupt.

enumerator kADC_wdg0HighThresholdIntEnable

Enable watchdog 0 high threshold interrupt.

enumerator kADC_wdg1LowThresholdIntEnable
Enable watchdog 1 low threshold interrupt.

enumerator kADC_wdg1HighThresholdIntEnable
Enable watchdog 1 high threshold interrupt.

enumerator kADC_wdg2LowThresholdIntEnable
Enable watchdog 2 low threshold interrupt.

enumerator kADC_wdg2HighThresholdIntEnable
Enable watchdog 2 high threshold interrupt.

enumerator kADC_wdg3LowThresholdIntEnable
Enable watchdog 3 low threshold interrupt.

enumerator kADC_wdg3HighThresholdIntEnable
Enable watchdog 3 high threshold interrupt.

enumerator kADC_wdg4LowThresholdIntEnable
Enable watchdog 4 low threshold interrupt.

enumerator kADC_wdg4HighThresholdIntEnable
Enable watchdog 4 high threshold interrupt.

enumerator kADC_wdg5LowThresholdIntEnable
Enable watchdog 5 low threshold interrupt.

enumerator kADC_wdg5HighThresholdIntEnable
Enable watchdog 5 high threshold interrupt.

enumerator kADC_wdg6LowThresholdIntEnable
Enable watchdog 6 low threshold interrupt.

enumerator kADC_wdg6HighThresholdIntEnable
Enable watchdog 6 high threshold interrupt.

enumerator kADC_wdg7LowThresholdIntEnable
Enable watchdog 7 low threshold interrupt.

enumerator kADC_wdg7HighThresholdIntEnable
Enable watchdog 7 high threshold interrupt.

enum _adc_self_test_int_enable
This enumeration provides the mask for the ADC self-test related interrupts enabling.

Values:

enumerator kADC_AlgSStep0ErrIntEnable
Enable self-test algorithm S step0 error interrupt.

enumerator kADC_AlgSStep1ErrIntEnable
Enable self-test algorithm S step1 error interrupt.

enumerator kADC_AlgSStep2ErrIntEnable
Enable self-test algorithm S step2 error interrupt.

enumerator kADC_AlgCErrIntEnable
Enable self-test algorithm C error interrupt.

enumerator kADC_AlgSEndIntEnable
Enable self-test algorithm S end interrupt.

enumerator kADC_AlgCEndIntEnable
Enable self-test algorithm C end interrupt.

enumerator kADC_ConvEndIntEnable
Enable self-test conversion end interrupt.

enumerator kADC_WdgTimeErrIntEnable
Enable watchdog time error interrupt.

enumerator kADC_WdgSequenceErrIntEnable
Enable watchdog sequence error interrupt.

enum _adc_conv_int_flag
This enumeration provides the mask for the ADC end-of-conversion and end-of-chain interrupts flag.

Values:

enumerator kADC_NormalConvChainEndIntFlag
Indicates whether the end of normal chain conversion interrupt has occurred.

enumerator kADC_NormalConvEndIntFlag
Indicates whether the end of conversion interrupt has occurred.

enumerator kADC_InjectConvChainEndIntFlag
Indicates whether the end of inject chain conversion interrupt has occurred.

enumerator kADC_InjectConvEndIntFlag
Indicates whether the end of inject conversion interrupt has occurred.

enumerator kADC_BctuConvEndIntFlag
Indicates whether the end of BCTU conversion interrupt has occurred.

enum _adc_wdg_threshold_int_flag
This enumeration provides the mask for the ADC analog watchdog threshold interrupts flag.

Values:

enumerator kADC_wdg0LowThresholdIntFlag
Indicates whether the watchdog 0 low threshold interrupt has occurred.

enumerator kADC_wdg0HighThresholdIntFlag
Indicates whether the watchdog 0 high threshold interrupt has occurred.

enumerator kADC_wdg1LowThresholdIntFlag
Indicates whether the watchdog 1 low threshold interrupt has occurred.

enumerator kADC_wdg1HighThresholdIntFlag
Indicates whether the watchdog 1 high threshold interrupt has occurred.

enumerator kADC_wdg2LowThresholdIntFlag
Indicates whether the watchdog 2 low threshold interrupt has occurred.

enumerator kADC_wdg2HighThresholdIntFlag
Indicates whether the watchdog 2 high threshold interrupt has occurred.

enumerator kADC_wdg3LowThresholdIntFlag
Indicates whether the watchdog 3 low threshold interrupt has occurred.

enumerator kADC_wdg3HighThresholdIntFlag
Indicates whether the watchdog 3 high threshold interrupt has occurred.

enumerator kADC_wdg4LowThresholdIntFlag
Indicates whether the watchdog 4 low threshold interrupt has occurred.

enumerator kADC_wdg4HighThresholdIntFlag
Indicates whether the watchdog 4 high threshold interrupt has occurred.

enumerator kADC_wdg5LowThresholdIntFlag
Indicates whether the watchdog 5 low threshold interrupt has occurred.

enumerator kADC_wdg5HighThresholdIntFlag
Indicates whether the watchdog 5 high threshold interrupt has occurred.

enumerator kADC_wdg6LowThresholdIntFlag
Indicates whether the watchdog 6 low threshold interrupt has occurred.

enumerator kADC_wdg6HighThresholdIntFlag
Indicates whether the watchdog 6 high threshold interrupt has occurred.

enumerator kADC_wdg7LowThresholdIntFlag
Indicates whether the watchdog 7 low threshold interrupt has occurred.

enumerator kADC_wdg7HighThresholdIntFlag
Indicates whether the watchdog 7 high threshold interrupt has occurred.

enum _adc_self_test_int_flag
This enumeration provides the mask for the ADC self-test-related interrupts flag.

Values:

enumerator kADC_AlgSStep0ErrIntFlag
Indicates whether the self-test algorithm S step0 error interrupt has occurred.

enumerator kADC_AlgSStep1ErrIntFlag
Indicates whether the self-test algorithm S step1 error interrupt has occurred.

enumerator kADC_AlgSStep2ErrIntFlag
Indicates whether the self-test algorithm S step2 error interrupt has occurred.

enumerator kADC_AlgCErrIntFlag
Indicates whether the self-test algorithm C error interrupt has occurred.

enumerator kADC_AlgSEndIntFlag
Indicates whether the algorithm S end interrupt has completed.

enumerator kADC_AlgCEndIntFlag
Indicates whether the algorithm C end interrupt has completed.

enumerator kADC_SelfTestConvEndIntFlag
Indicates whether the self-test end-of-conversion interrupt has completed.

enumerator kADC_OverWriteErrIntFlag
Indicates whether the overwrite error interrupt has occurred.

enumerator kADC_WdgTimeErrIntFlag
Indicates whether the watchdog time error interrupt has occurred.

enumerator kADC_WdgSequenceErrIntFlag
Indicates whether the watchdog sequence error interrupt has occurred.

enum _adc_bctu_mode
This enumeration provides the selection of the Body Cross Trigger Unit (BCTU) mode.

Values:

enumerator kADC_BctuModeDisable
BCTU disabled.

enumerator kADC_BctuTrig
BCTU enabled, only BCTU can trigger conversion.

enumerator kADC_AllTrig
BCTU enabled, all trigger sources can trigger conversion.

enum _adc_conv_res
This enumeration provides the selection of the ADC conversion resolution.

Values:

enumerator kADC_ConvRes14Bit
14-bit resolution.

enumerator kADC_ConvRes12Bit
12-bit resolution.

enumerator kADC_ConvRes10Bit
10-bit resolution.

enumerator kADC_ConvRes8Bit
8-bit resolution.

enum _adc_ext_trig
This enumeration provides the selection of the ADC external trigger type.

Values:

enumerator kADC_ExtTrigDisable
Normal trigger input does not start a conversion.

enumerator kADC_ExtTrigFallingEdge
Normal trigger (falling edge) input starts a conversion.

enumerator kADC_ExtTrigRisingEdge
Normal trigger (rising edge) input starts a conversion.

enum _adc_speed_mode
This enumeration provides the selection of the ADC conversion speed mode.

Values:

enumerator kADC_SpeedModeNormal
Normal conversion speed.

enumerator kADC_SpeedModeHigh
High-speed conversion.

enum _adc_conv_avg
This enumeration provides the selection of the number of conversions ADC uses to calculate the conversion result.

Values:

enumerator kADC_ConvAvgDisable
Conversions averaging disabled.

enumerator kADC_ConvAvg4
4 conversions averaging.

enumerator kADC_ConvAvg8
8 conversions averaging.

enumerator kADC_ConvAvg16
16 conversions averaging.

enumerator kADC_ConvAvg32
32 conversions averaging.

enum _adc_state
This enumeration provides the selection of the ADC state.

Values:

enumerator kADC_AdcIdle
Indicates the ADC is in the IDLE state.

enumerator kADC_AdcPowerdown
Indicates the ADC is in the power-down state.

enumerator kADC_AdcWait
Indicates the ADC is in the wait state.

enumerator kADC_AdcBusyInCalibration
Indicates the ADC is in the calibration busy state.

enumerator kADC_AdcSample
Indicates the ADC is in the sample state.

enumerator kADC_AdcConv
Indicates the ADC is in the conversion state.

enum _adc_conv_mode
This enumeration provides the selection of the ADC conversion mode, including normal conversion one-shot mode, normal conversion scan mode, and inject conversion one-shot mode.

Values:

enumerator kADC_NormalConvOneShotMode
Normal conversion one-shot mode.

enumerator kADC_NormalConvScanMode
Normal conversion scan mode.

enumerator kADC_InjectConvOneShotMode
Inject conversion one-shot mode.

enum _adc_clock_frequency
This enumeration provides the selection of the ADC operating clock frequency, including half-bus frequency and full bus frequency.

Values:

enumerator kADC_HalfBusFrequency
Half of bus clock frequency.

enumerator kADC_FullBusFrequency
Equal to bus clock frequency.

enumerator kADC_ModuleClockFreq
Module clock frequency.

enumerator kADC_ModuleClockFreqDivide2
Module clock frequency / 2.

enumerator kADC_ModuleClockFreqDivide4
Module clock frequency / 4.

enumerator kADC_ModuleClockFreqDivide8
Module clock frequency / 8.

enum _adc_conv_data_align
This enumeration provides the selection of the ADC conversion data alignment, including the right alignment and left alignment.

Values:

enumerator kADC_ConvDataRightAlign
Conversion data is right aligned.

enumerator kADC_ConvDataLeftAlign
Conversion data is left aligned.

enum _adc_presample_voltage_src
This enumeration provides the selection of the ADC internal analog input voltage sources for pre-sample, including DVDD0P8/2, AVDD1P8/4, VREFL_1p8 and VREFH_1p8.

Values:

enumerator kADC_PresampleVoltageSrcVREL
Use VREL as pre-sample voltage source.

enumerator kADC_PresampleVoltageSrcVREH
Use VREH as pre-sample voltage source.

enum _adc_dma_request_clear_src
This enumeration provides the selection of the DMA request clear sources, including clear by acknowledgment from the DMA controller and clear on a read of the data register.

Values:

enumerator kADC_DMARequestClearByAck
DMA request cleared by acknowledgment from DMA controller.

enumerator kADC_DMARequestClearOnRead
DMA request cleared on a read of the data register.

enum _adc_average_sample_numbers
This enumeration provides the selection of the ADC calibration averaging sample numbers, including 16, 32, 128, and 512 averaging samples.

Values:

enumerator kADC_AverageSampleNumbers4
Use 4 averaging samples during calibration.

enumerator kADC_AverageSampleNumbers8
Use 8 averaging samples during calibration.

enumerator kADC_AverageSampleNumbers16
Use 16 averaging samples during calibration.

enumerator kADC_AverageSampleNumbers32
Use 32 averaging samples during calibration.

enum _adc_sample_time

This enumeration provides the selection of the ADC sample time of calibration conversions, including 22, 8, 16 and 32 cycles of ADC_CLK.

Values:

enumerator kADC_SampleTime22

Use 22 cycles of ADC_CLK as sample time of calibration conversions.

enumerator kADC_SampleTime8

Use 8 cycles of ADC_CLK as sample time of calibration conversions.

enumerator kADC_SampleTime16

Use 16 cycles of ADC_CLK as sample time of calibration conversions.

enumerator kADC_SampleTime32

Use 32 cycles of ADC_CLK as sample time of calibration conversions.

enum _adc_wdg_threshold_int

This enumeration provides the selection of the ADC analog watchdog threshold low and high interrupt enable.

Values:

enumerator kADC_LowHighThresholdIntDisable

Enable the ADC analog watchdog low and high threshold interrupts.

enumerator kADC_LowThresholdIntEnable

Enable the ADC analog watchdog low threshold interrupt.

enumerator kADC_HighThresholdIntEnable

Enable the ADC analog watchdog high threshold interrupt.

enumerator kADC_LowHighThresholdIntEnable

Enable the ADC analog watchdog low and high threshold interrupts.

enum _adc_alg_type

This enumeration provides the selection of the ADC self-test algorithm type, including algorithm S, algorithm C and algorithm S and C.

Note: The meaning of enumeration member 'kADC_SelfTestForAlgSAndC' in different conversion modes is different, in the one-shot conversion mode, it means executing algorithm S; in the scan conversion mode, it means executing algorithm S and C.

Values:

enumerator kADC_SelfTestForAlgS

Use algorithm S for self-test.

enumerator kADC_SelfTestForAlgC

Use algorithm C for self-test.

enumerator kADC_SelfTestForAlgSAndC

Use algorithm S for one-shot conversion mode self-test, use algorithm S and algorithm C for scan conversion mode self-test.

enum _adc_self_test_wdg_threshold

This enumeration provides the selection of the ADC self-test watchdog thresholds for algorithm S step 0 - 2, and watchdog thresholds for algorithm C step 0 and step x (x = 1 - 11).

Values:

enumerator kADC_SelfTestWdgThresholdForAlgSStep0
Self-test watchdog threshold for the algorithm S step 0.

enumerator kADC_SelfTestWdgThresholdForAlgSStep1
Self-test watchdog threshold for the algorithm S step 1 fraction part.

enumerator kADC_SelfTestWdgThresholdForAlgSStep2
Self-test watchdog threshold for the algorithm S step 2.

enumerator kADC_SelfTestWdgThresholdForAlgCStep0
Self-test watchdog threshold for the algorithm C step 0.

enumerator kADC_SelfTestWdgThresholdForAlgCStepx
Self-test watchdog threshold for the algorithm C step x.

enum _adc_wdg_timer_val
This enumeration provides the selection of the ADC self-test watchdog timer value, including 0.1ms, 0.5ms, 1ms, 2ms, 5ms, 10ms, 20ms and 50ms.

Values:

enumerator kADC_SelfTestWdgTimerVal0
0.1ms ((0008h × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal1
0.5ms ((0027h × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal2
1ms ((004Eh × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal3
2ms ((009Ch × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal4
5ms ((0187h × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal5
10ms ((030Dh × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal6
20ms ((061Ah × Prescaler) cycles at 80 MHz).

enumerator kADC_SelfTestWdgTimerVal7
50ms ((0F42h × Prescaler) cycles at 80 MHz).

typedef enum *_adc_bctu_mode* adc_bctu_mode_t
This enumeration provides the selection of the Body Cross Trigger Unit (BCTU) mode.

typedef enum *_adc_conv_res* adc_conv_res_t
This enumeration provides the selection of the ADC conversion resolution.

typedef enum *_adc_ext_trig* adc_ext_trig_t
This enumeration provides the selection of the ADC external trigger type.

typedef enum *_adc_speed_mode* adc_speed_mode_t
This enumeration provides the selection of the ADC conversion speed mode.

typedef enum *_adc_conv_avg* adc_conv_avg_t
This enumeration provides the selection of the number of conversions ADC uses to calculate the conversion result.

typedef enum *_adc_state* adc_state_t
This enumeration provides the selection of the ADC state.

typedef enum *_adc_conv_mode* adc_conv_mode_t

This enumeration provides the selection of the ADC conversion mode, including normal conversion one-shot mode, normal conversion scan mode, and inject conversion one-shot mode.

typedef enum *_adc_clock_frequency* adc_clock_frequency_t

This enumeration provides the selection of the ADC operating clock frequency, including half-bus frequency and full bus frequency.

typedef enum *_adc_conv_data_align* adc_conv_data_align_t

This enumeration provides the selection of the ADC conversion data alignment, including the right alignment and left alignment.

typedef enum *_adc_presample_voltage_src* adc_presample_voltage_src_t

This enumeration provides the selection of the ADC internal analog input voltage sources for pre-sample, including DVDD0P8/2, AVDD1P8/4, VREFL_1p8 and VREFH_1p8.

typedef enum *_adc_dma_request_clear_src* adc_dma_request_clear_src_t

This enumeration provides the selection of the DMA request clear sources, including clear by acknowledgment from the DMA controller and clear on a read of the data register.

typedef enum *_adc_average_sample_numbers* adc_average_sample_numbers_t

This enumeration provides the selection of the ADC calibration averaging sample numbers, including 16, 32, 128, and 512 averaging samples.

typedef enum *_adc_sample_time* adc_sample_time_t

This enumeration provides the selection of the ADC sample time of calibration conversions, including 22, 8, 16 and 32 cycles of ADC_CLK.

typedef enum *_adc_wdg_threshold_int* adc_wdg_threshold_int_t

This enumeration provides the selection of the ADC analog watchdog threshold low and high interrupt enable.

typedef enum *_adc_alg_type* adc_alg_type_t

This enumeration provides the selection of the ADC self-test algorithm type, including algorithm S, algorithm C and algorithm S and C.

Note: The meaning of enumeration member 'kADC_SelfTestForAlgSAndC' in different conversion modes is different, in the one-shot conversion mode, it means executing algorithm S; in the scan conversion mode, it means executing algorithm S and C.

---

typedef enum *_adc_self_test_wdg_threshold* adc_self_test_wdg_threshold_t

This enumeration provides the selection of the ADC self-test watchdog thresholds for algorithm S step 0 - 2, and watchdog thresholds for algorithm C step 0 and step x (x = 1 - 11).

typedef enum *_adc_wdg_timer_val* adc_wdg_timer_val_t

This enumeration provides the selection of the ADC self-test watchdog timer value, including 0.1ms, 0.5ms, 1ms, 2ms, 5ms, 10ms, 20ms and 50ms.

typedef struct *_adc_config* adc_config_t

This structure is used to configure the ADC module.

typedef struct *_adc_channel_config* adc_channel_config_t

This structure is used to configure the ADC conversion channel.

typedef struct *_adc_chain_config* adc_chain_config_t

This structure is used to configure the ADC conversion chain.

typedef struct *_adc_wdg_config* adc_wdg_config_t

This structure is used to configure the ADC analog watchdog.

typedef struct *_adc_calibration_config* adc_calibration_config_t

This structure is used to configure the ADC calibration.

typedef struct *_adc_user_offset_gain_config* adc_user_offset_gain_config_t

This structure is used to configure the ADC user offset and gain.

typedef struct *_adc_self_test_config* adc_self_test_config_t

This structure is used to configure the ADC self-test.

typedef struct *_adc_self_test_wdg_config* adc_self_test_wdg_config_t

This structure is used to configure the ADC self-test watchdog for algorithm steps.

Note: The algorithm S step 2 only has the 'LowThrsholdVal'.

typedef struct *_adc_conv_result* adc_conv_result_t

This structure is used to save the result information when obtaining the conversion result.

typedef struct *_adc_self_test_conv_result* adc_self_test_conv_result_t

This structure is used to save the result information when obtaining the self-test channel conversion result.

Note: The member 'convData' is used to store self-test channel conversion results. Only when executing step 1 of algorithm S, member 'convDataFraction' will be used to store the fractional part data. When executing other algorithms, this member will not be used.

ADC_GROUP_COUNTS

ADC_THRESHOLD_COUNTS

ADC_SELF_TEST_THRESHOLD_COUNTS

GET_REGINDEX(channelIndex)

GET_BITINDEX(channelIndex)

REGISTER_READWRITE(baseRegister, shiftIndex)

REGISTER_READONLY(baseRegister, shiftIndex)

NCMR_IO(base, registerIndex)

JCMR_IO(base, registerIndex)

PSR_IO(base, registerIndex)

DMAR_IO(base, registerIndex)

CWSELR_IO(base, registerIndex)

CWENR_IO(base, registerIndex)

CIMR_IO(base, registerIndex)

CEOCFR_IO(base, registerIndex)

AWORR_IO(base, registerIndex)

STAWR_IO(base, registerIndex)

CEOCFR_I(base, registerIndex)

AWORR_I(base, registerIndex)

CDR_I(base, registerIndex)

WDG_SELECT_MASK(shiftIndex)

WDG_SELECT_SHIFT(shiftIndex)

WDG_SELECT(val, shiftIndex)

ADC_CDR_VALID_MASK

ADC_CDR_VALID_SHIFT

ADC_CDR_OVERW_MASK

ADC_CDR_OVERW_SHIFT

ADC_CDR_RESULT_MASK

ADC_CDR_RESULT_SHIFT

ADC_CDR_CDATA_MASK

ADC_CDR_CDATA_SHIFT

ADC_STAWR_AWDE_MASK

ADC_STAWR_THRL_MASK

ADC_STAWR_THRL_SHIFT

ADC_STAWR_THRL(val)

ADC_STAWR_THRH_MASK

ADC_STAWR_THRH_SHIFT

ADC_STAWR_THRH(val)

ADC_CALSTAT_MAX

ADC_CALSTAT_SIGN

struct _adc_config

#include <fsl_sar_adc.h> This structure is used to configure the ADC module.

Public Members

bool enableAutoClockOff

Decides whether to enable the ADC auto clock-off function, when set to true, the internal ADC clock is automatically switched off during IDLE mode to reduce power consumption (without going into power-down mode).

bool enableOverWrite

Decides whether to enable the latest conversion to overwrite the current value in the data registers.

bool enableConvertPresampleVal

Decides whether to convert the pre-sampled value, if enabled, pre-sampling is followed by the conversion, sampling will be bypassed and conversion of the pre-sampled data will be done.

adc_speed_mode_t speedMode
Selects ADC speed mode.

uint16_t convDelay
Specifies the delay in terms of the number of module clock cycles. In case the channel to convert changed since the last conversion and this new channel is an external channel, the conversion starts after a delay configured by convDelay.

adc_bctu_mode_t bctuMode
Selects the BCTU mode.

adc_conv_res_t convRes
Specifies the number of significant bits per conversion data.

adc_conv_avg_t convAvg
Selects the number of conversions ADC uses to calculate the conversion result.

adc_ext_trig_t extTrig
Specifies whether the normal trigger (with trigger type) input starts a conversion.

adc_conv_data_align_t convDataAlign
Selects the conversion data alignment.

adc_clock_frequency_t clockFrequency
Selects the ADC clock frequency.

adc_dma_request_clear_src_t dmaRequestClearSrc
Selects DMA request clear source.

adc_presample_voltage_src_t presampleVoltageSrc[1]
Selects analog input voltages for group 0 (corresponding to channel 0 to channel 31) and group 32 (corresponding to channel 32 to channel 63) pre-sampling.

uint8_t samplePhaseDuration[1]
Sets the sample phase duration in terms of the ADC controller clock for group 0 (corresponding to channel 0 to channel 31) and group 32 (corresponding to channel 32 to channel 63), the minimum acceptable value is 8, configuring to a value lower than 8 sets the sample period to 8 cycles.

struct __adc__channel__config
#include <fsl_sar_adc.h> This structure is used to configure the ADC conversion channel.

Public Members

uint8_t channelIndex
Sets the conversion channel index.

bool enableInt
Decides Whether to enable the interrupt function of the current conversion channel.

bool enablePresample
Decides whether to enable the pre-sample function of the current conversion channel.

bool enableDmaTransfer
Decides whether to enable the DMA transfer function of the current conversion channel.

bool enableWdg
Decides whether to enable the analog watchdog function of the current conversion channel.

uint8_t wdgIndex
Indicates which analog watchdog to provide the low and high threshold value.

struct _adc_chain_config
#include <fsl_sar_adc.h> This structure is used to configure the ADC conversion chain.

Public Members

adc_conv_mode_t convMode
Selects conversion mode.

bool enableGlobalChannelConvEndInt
Global control function to determine whether to enable the interrupt function of conversion channels.

bool enableChainConvEndInt
Decides whether to enable the current chain end-of-conversion interrupt.

uint8_t channelCount
Indicates the channel counts.

adc_channel_config_t *channelConfig
Chain channels configuration.

struct _adc_wdg_config
#include <fsl_sar_adc.h> This structure is used to configure the ADC analog watchdog.

Public Members

uint8_t wdgIndex
Indicates the analog watchdog index

adc_wdg_threshold_int_t wdgThresholdInt
Selects watchdog threshold low or/and high interrupt to enable/disable.

uint16_t lowThresholdVal
Sets the ADC analog watchdog low threshold value.

uint16_t highThresholdVal
Sets the ADC analog watchdog high threshold value.

struct _adc_calibration_config
#include <fsl_sar_adc.h> This structure is used to configure the ADC calibration.

Public Members

bool enableAverage
Decides whether to enable averaging of calibration time.

adc_sample_time_t sampleTime
Selects sample time of calibration conversions.

adc_average_sample_numbers_t averageSampleNumbers
Selects calibration averaging sample numbers.

struct _adc_user_offset_gain_config
#include <fsl_sar_adc.h> This structure is used to configure the ADC user offset and gain.

Public Members

int8_t userOffset

Sets user defined gain value.

int16_t userGain

Sets user defined offset value.

struct __adc__self__test__config

#include <fsl_sar_adc.h> This structure is used to configure the ADC self-test.

Public Members

adc_alg_type_t algType

Selects the self-test algorithm.

uint8_t algSteps

Sets the self-test algorithm steps, it should be programmed to zero in scan mode.

uint8_t algSSamplePhaseDuration

Sets the self-test algorithm S conversion sampling phase duration.

uint8_t algCSamplePhaseDuration

Sets the self-test algorithm C conversion sampling phase duration.

uint8_t baudRate

Sets the baud rate for the selected algorithm in scan mode, must write to this field before enabling a self-test channel.

struct __adc__self__test__wdg__config

#include <fsl_sar_adc.h> This structure is used to configure the ADC self-test watchdog for algorithm steps.

Note: The algorithm S step 2 only has the 'LowThrsholdVal'.

Public Members

adc_self_test_wdg_threshold_t wdgThresholdId

Indicates the self-test watchdog index

uint16_t lowThrsholdVal

Sets the self-test watchdog low threshold value.

uint16_t highThrsholdVal

Sets the self-test watchdog high threshold value.

struct __adc__conv__result

#include <fsl_sar_adc.h> This structure is used to save the result information when obtaining the conversion result.

Public Members

bool overWrittenFlag

Indicates when conversion data was overwritten by a newer result, the new data is written or discarded according to MCR[OWREN].

uint8_t convMode
Indicates the mode of conversion for the corresponding channel.

uint16_t convData
Stores the conversion data corresponding to the internal channel.

struct __adc__self__test__conv__result
#include <fsl_sar_adc.h> This structure is used to save the result information when obtaining the self-test channel conversion result.

Note: The member 'convData' is used to store self-test channel conversion results. Only when executing step 1 of algorithm S, member 'convDataFraction' will be used to store the fractional part data. When executing other algorithms, this member will not be used.

Public Members

bool overWrittenFlag
Indicates when conversion data is overwritten by a newer result.

uint16_t convData
Stores the conversion data corresponding to the internal self-test channel.

uint16_t convDataFraction
This field is only used to store the fractional part conversion result.

2.59 SEMA42: Hardware Semaphores Driver

FSL__SEMA42__DRIVER__VERSION
SEMA42 driver version.

SEMA42 status return codes.

Values:

enumerator kStatus__SEMA42__Busy
SEMA42 gate has been locked by other processor.

enumerator kStatus__SEMA42__Reseting
SEMA42 gate reseting is ongoing.

enum __sema42__gate__status
SEMA42 gate lock status.

Values:

enumerator kSEMA42__Unlocked
The gate is unlocked.

enumerator kSEMA42__LockedByProc0
The gate is locked by processor 0.

enumerator kSEMA42__LockedByProc1
The gate is locked by processor 1.

enumerator kSEMA42__LockedByProc2
The gate is locked by processor 2.

enumerator kSEMA42_LockedByProc3
The gate is locked by processor 3.

enumerator kSEMA42_LockedByProc4
The gate is locked by processor 4.

enumerator kSEMA42_LockedByProc5
The gate is locked by processor 5.

enumerator kSEMA42_LockedByProc6
The gate is locked by processor 6.

enumerator kSEMA42_LockedByProc7
The gate is locked by processor 7.

enumerator kSEMA42_LockedByProc8
The gate is locked by processor 8.

enumerator kSEMA42_LockedByProc9
The gate is locked by processor 9.

enumerator kSEMA42_LockedByProc10
The gate is locked by processor 10.

enumerator kSEMA42_LockedByProc11
The gate is locked by processor 11.

enumerator kSEMA42_LockedByProc12
The gate is locked by processor 12.

enumerator kSEMA42_LockedByProc13
The gate is locked by processor 13.

enumerator kSEMA42_LockedByProc14
The gate is locked by processor 14.

typedef enum *_sema42_gate_status* sema42_gate_status_t
SEMA42 gate lock status.

void SEMA42_Init(SEMA42_Type *base)
Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42_ResetGate or SEMA42_ResetAllGates function.

Parameters

- base – SEMA42 peripheral base address.

void SEMA42_Deinit(SEMA42_Type *base)
De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

- base – SEMA42 peripheral base address.

status_t SEMA42_TryLock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)
Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_SEMA42_Busy – Sema42 gate has been locked by another processor.

status_t SEMA42_Lock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns

status_t

static inline void SEMA42_Unlock(SEMA42_Type *base, uint8_t gateNum)

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline *sema42_gate_status_t* SEMA42_GetGateStatus(SEMA42_Type *base, uint8_t gateNum)

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Returns

status Current status.

status_t SEMA42_ResetGate(SEMA42_Type *base, uint8_t gateNum)

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – SEMA42 gate is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

static inline *status_t* SEMA42_ResetAllGates(SEMA42_Type *base)

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.

Return values

- kStatus_Success – SEMA42 is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

SEMA42_GATE_NUM_RESET_ALL

The number to reset all SEMA42 gates.

SEMA42_GATEn(base, n)

SEMA42 gate n register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42_GATEn gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U: 0 ^ 3 = 3 1 ^ 3 = 2 2 ^ 3 = 1 3 ^ 3 = 0 4 ^ 3 = 7 5 ^ 3 = 6 6 ^ 3 = 5 7 ^ 3 = 4 ...

SEMA42_BUSY_POLL_COUNT

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.60 Siul2

FSL_SIUL2_DRIVER_VERSION

SIUL2 driver version.

FEATURE_SIUL2_MAX_NUMBER_OF_INPUT

SIUL2 module maximum number of input signal on a pin.

FEATURE_ADC_INTERLEAVE_MAX_MUX_MODE

Some pins have two ADC interleave config, such as GPIO45 can be muxed as ADC0_S8 & ADC2_S8.

DCM_DCMRWF4_ADC_INTERLEAVE_MASK

Mask all adc interleave bits, some bit may not exist on some parts.

SIUL2_PORT_WRITE8(address, value)

SIUL2_PORT_WRITE32(address, value)

SIUL2_PORT_READ32(address)

SIUL2_PORT_READ8(address)

SIUL2_GPDO_ADDR(base, pin)

SIUL2_GPDI_ADDR(base, pin)

SIUL2_PGPDO_REG_16_31(base, x)

SIUL2_PGPDO_REG_0_15(base, x)

enum siul2_port_pull_config

Internal resistor pull feature selection.

Values:

enumerator kPORT_INTERNAL_PULL_DOWN_ENABLED

internal pull-down resistor is enabled.

enumerator kPORT_INTERNAL_PULL_UP_ENABLED

internal pull-up resistor is enabled.

enumerator kPORT_INTERNAL_PULL_NOT_ENABLED

internal pull-down/up resistor is disabled.

enum siul2_port_mux

Configures the Pin output muxing selection.

Values:

enumerator kPORT_MUX_AS_GPIO

corresponding pin is configured as GPIO

enumerator kPORT_MUX_ALT1

chip-specific

enumerator kPORT_MUX_ALT2

chip-specific

enumerator kPORT_MUX_ALT3

chip-specific

enumerator kPORT_MUX_ALT4

chip-specific

enumerator kPORT_MUX_ALT5

chip-specific

enumerator kPORT_MUX_ALT6

chip-specific

enumerator kPORT_MUX_ALT7
chip-specific

enumerator kPORT_MUX_ALT8
chip-specific

enumerator kPORT_MUX_ALT9
chip-specific

enumerator kPORT_MUX_ALT10
chip-specific

enumerator kPORT_MUX_ALT11
chip-specific

enumerator kPORT_MUX_ALT12
chip-specific

enumerator kPORT_MUX_ALT13
chip-specific

enumerator kPORT_MUX_ALT14
chip-specific

enumerator kPORT_MUX_ALT15
chip-specific

enumerator kPORT_MUX_NOT_AVAILABLE
chip-specific

enum siul2_port_input_filter
Configures the Pin filter enable.

Values:

enumerator kPORT_INPUT_FILTER_DISABLED
IFE OFF

enumerator kPORT_INPUT_FILTER_ENABLED
IFE ON

enumerator kPORT_INPUT_FILTER_NOT_AVAILABLE
IFE NOT AVAILABLE

enum port_pull_keep
Configures the Pad keep enable.

Values:

enumerator kPORT_PULL_KEEP_DISABLED
PKE OFF

enumerator kPORT_PULL_KEEP_ENABLED
PKE ON

enumerator kPORT_PULL_KEEP_NOT_AVAILABLE
PKE NOT AVAILABLE

enum siul2_port_invert
Configures signal invert for the pin.

Values:

enumerator kPORT_INVERT_DISABLED
INV OFF

enumerator kPORT_INVERT_ENABLED
INV ON

enumerator kPORT_INVERT_NOT_AVAILABLE
INV NOT AVAILABLE

enum siul2_port_output_buffer
Configures the output buffer enable.

Values:

enumerator kPORT_OUTPUT_BUFFER_DISABLED
Output buffer disabled

enumerator kPORT_OUTPUT_BUFFER_ENABLED
Output buffer enabled

enumerator kPORT_OUTPUT_BUFFER_NOT_AVAILABLE
Output buffer not available

enum siul2_port_input_buffer
Configures the Input Buffer Enable field.

Values:

enumerator kPORT_INPUT_BUFFER_DISABLED
Input buffer disabled

enumerator kPORT_INPUT_BUFFER_ENABLED
Input buffer enabled

enumerator kPORT_INPUT_BUFFER_NOT_AVAILABLE
Input buffer not available

enum siul2_port_inputmux_t
Configures the Pin input muxing selection.

Values:

enumerator kPORT_INPUT_MUX_ALT0
Chip-specific

enumerator kPORT_INPUT_MUX_ALT1
Chip-specific

enumerator kPORT_INPUT_MUX_ALT2
Chip-specific

enumerator kPORT_INPUT_MUX_ALT3
Chip-specific

enumerator kPORT_INPUT_MUX_ALT4
Chip-specific

enumerator kPORT_INPUT_MUX_ALT5
Chip-specific

enumerator kPORT_INPUT_MUX_ALT6
Chip-specific

enumerator kPORT_INPUT_MUX_ALT7
Chip-specific

enumerator kPORT_INPUT_MUX_ALT8
Chip-specific

enumerator kPORT_INPUT_MUX_ALT9
Chip-specific

enumerator kPORT_INPUT_MUX_ALT10
Chip-specific

enumerator kPORT_INPUT_MUX_ALT11
Chip-specific

enumerator kPORT_INPUT_MUX_ALT12
Chip-specific

enumerator kPORT_INPUT_MUX_ALT13
Chip-specific

enumerator kPORT_INPUT_MUX_ALT14
Chip-specific

enumerator kPORT_INPUT_MUX_ALT15
Chip-specific

enumerator kPORT_INPUT_MUX_NO_INIT
No initialization

enum siul2_port_safe_mode
Configures the Safe Mode Control.

Values:

enumerator kPORT_SAFE_MODE_DISABLED
To drive pad in hi-z state using OBE = 0, when FCCU in fault state. The OBE will be driven by IP/SIUL when FCCU leaves the fault state.

enumerator kPORT_SAFE_MODE_ENABLED
No effect on IP/SIUL driven OBE value

enumerator kPORT_SAFE_MODE_NOT_AVAILABLE
Not available

enum siul2___port_slew_rate
Configures the slew rate control.

Values:

enumerator kPORT_SLEW_RATE_FASTEST
Fmax=133 MHz(at 1.8V), 100 MHz (at 3.3V), apply for SIUL2_0/1

enumerator kPORT_SLEW_RATE_SLOWEST
Fmax=83 MHz (at 1.8V), 63 MHz (at 3.3V), apply for SIUL2_0/1

enumerator kPORT_SLEW_RATE_NOT_AVAILABLE
Not available

enum siul2_port_drive_strength
Configures the drive strength.

Values:

enumerator kPORT_DRIVE_STRENTGTH_DISABLED
Disables DSE.

enumerator kPORT_DRIVE_STRENTGTH_ENABLED
Enables DSE.

enumerator kPORT_DRIVE_STRENTGTH_NOT_AVAILABLE
Not available.

enum siul2_port_direction
Configures port direction.

Values:

enumerator kPORT_IN
Sets port pin as input.

enumerator kPORT_OUT
Sets port pin as output.

enumerator kPORT_IN_OUT
Sets port pin as bidirectional.

enumerator kPORT_HI_Z
Sets port pin as high_z.

enum siul2_adc_interleaves
Configures adc interleave mux mode. Note! Not all are supported for a given part, please refer to IOMUX table for supported interleaves.

Values:

enumerator kMUX_MODE_NOT_AVAILABLE
Adc Interleave not available.

enumerator kMUX_MODE_EN_ADC1_S18_1
Set bit ADC1_S18 to 1

enumerator kMUX_MODE_EN_ADC0_S8_1
Set bit ADC0_S8 to 1

enumerator kMUX_MODE_EN_ADC0_S9_1
Set bit ADC0_S9 to 1

enumerator kMUX_MODE_EN_ADC1_S14_1
Set bit ADC1_S14 to 1

enumerator kMUX_MODE_EN_ADC1_S15_1
Set bit ADC1_S15 to 1

enumerator kMUX_MODE_EN_ADC1_S22_1
Set bit ADC1_S22 to 1

enumerator kMUX_MODE_EN_ADC1_S23_1
Set bit ADC1_S23 to 1

enumerator kMUX_MODE_EN_ADC0_S12_1
Set bit ADC0_S12 to 1

enumerator kMUX_MODE_EN_ADC0_S13_1
Set bit ADC0_S13 to 1

enumerator kMUX_MODE_EN_ADC2_S8_1
Set bit ADC2_S8 to 1

enumerator kMUX_MODE_EN_ADC2_S9_1
Set bit ADC2_S9 to 1

enumerator kMUX_MODE_EN_ADC0_S14_1
Set bit ADC0_S14 to 1

enumerator kMUX_MODE_EN_ADC0_S17_1
Set bit ADC0_S17 to 1

enumerator kMUX_MODE_EN_ADC1_S18_0
With bits 15-0, only clear ADC1_S18 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC0_S8_0
With bits 15-0, only clear ADC0_S8 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC0_S9_0
With bits 15-0, only clear ADC0_S9 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC1_S14_0
With bits 15-0, only clear ADC1_S14 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC1_S15_0
With bits 15-0, only clear ADC1_S15 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC1_S22_0
With bits 15-0, only clear ADC1_S22 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC1_S23_0
With bits 15-0, only clear ADC1_S23 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC0_S12_0
With bits 15-0, only clear ADC0_S12 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC0_S13_0
With bits 15-0, only clear ADC0_S13 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC2_S8_0
With bits 15-0, only clear ADC2_S8 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC2_S9_0
With bits 15-0, only clear ADC2_S9 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC0_S14_0
With bits 15-0, only clear ADC0_S14 bit, the other bits set to 1

enumerator kMUX_MODE_EN_ADC0_S17_0
With bits 15-0, only clear ADC0_S17 bit, the other bits set to 1

enum _siul2_port_num
PORT definition.

Values:

enumerator kSIUL2_PTA
PTA.

enumerator kSIUL2_PTB
PTB.

enumerator kSIUL2_PTC
PTC.

enumerator kSIUL2_PTD
PTD.

enumerator kSIUL2_PTE
PTE.

enumerator kSIUL2_PTF
PTF.

enumerator kSIUL2_PTG
PTG.

enum siul2_interrupt_config
SIUL2 Interrupt Configuration.

This structure is used to configure the interrupt.

Values:

enumerator kSIUL2_InterruptStatusFlagDisabled
Interrupt status flag is disabled.

enumerator kSIUL2_InterruptRisingEdge
Interrupt on rising edge.

enumerator kSIUL2_InterruptFallingEdge
Interrupt on falling edge.

enumerator kSIUL2_InterruptBothEdge
Interrupt on both edges.

typedef uint8_t siul2_port_pins_Level_t
Type of a port levels representation.

typedef enum *siul2_port_pull_config* siul2_port_pull_config_t
Internal resistor pull feature selection.

typedef enum *siul2_port_mux* siul2_port_mux_t
Configures the Pin output muxing selection.

typedef enum *siul2_port_input_filter* siul2_port_input_filter_t
Configures the Pin filter enable.

typedef enum *port_pull_keep* siul2_port_pull_keep_t
Configures the Pad keep enable.

typedef enum *siul2_port_invert* siul2_port_invert_t
Configures signal invert for the pin.

typedef enum *siul2_port_output_buffer* siul2_port_output_buffer_t
Configures the output buffer enable.

typedef enum *siul2_port_input_buffer* siul2_port_input_buffer_t
Configures the Input Buffer Enable field.

typedef enum *siul2_port_safe_mode* siul2_port_safe_mode_t
Configures the Safe Mode Control.

typedef enum *siul2_port_slew_rate* siul2_port_slew_rate_t
Configures the slew rate control.

typedef enum *siul2_port_drive_strength* siul2_port_drive_strength_t
Configures the drive strength.

typedef enum *siul2_port_direction* siul2_port_direction_t

Configures port direction.

typedef enum *siul2_adc_interleaves* siul2_adc_interleaves_t

Configures adc interleave mux mode. Note! Not all are supported for a given part, please refer to IOMUX table for supported interleaves.

typedef struct *siul2_pin_settings* siul2_pin_settings_t

Defines the converter configuration.

This structure is used to configure the pins

typedef enum *_siul2_port_num* siul2_port_num_t

PORT definition.

typedef struct *siul2_filter_config* siul2_filter_config_t

SIUL2 Interrupt Filter Configuration.

This structure is used to configure the filter for interrupt.

typedef enum *siul2_interrupt_config* siul2_interrupt_config_t

SIUL2 Interrupt Configuration.

This structure is used to configure the interrupt.

typedef void (*siul2_cb_t)(SIUL2_Type *base, uint32_t status)

void SIUL2_PinInit(const *siul2_pin_settings_t* *config)

Initialize pin.

Parameters

- config – the pin's setting siul2_pin_settings_t.

void SIUL2_SetPinInputBuffer(SIUL2_Type *base, uint32_t pin, bool enable, uint32_t inputMuxReg, *siul2_port_inputmux_t* inputMux)

Set the pin Input Buffer.

Parameters

- base – SIUL2 peripheral base pointer
- pin – pin number, 0, 1...511, see RM for available pins
- enable – Enable input buffer
- inputMuxReg – Pin muxing register slot selection
- inputMux – Pin muxing slot selection

void SIUL2_SetPinOutputBuffer(SIUL2_Type *base, uint32_t pin, bool enable, *siul2_port_mux_t* mux)

Set the pin Output Buffer.

Parameters

- base – SIUL2 peripheral base pointer
- pin – pin number, 0, 1...511, see RM for available pins
- enable – Enable output buffer
- mux – Pin muxing slot selection

void SIUL2_SetPinPullSel(SIUL2_Type *base, uint32_t pin, *siul2_port_pull_config_t* pullConfig)

Configures the pin pull select.

This function configures the internal resistor.

Parameters

- base – SIUL2 peripheral base pointer
- pin – pin number, 0, 1...511, see RM for available pins
- pullConfig – The pull configuration

void SIUL2_SetPinDirection(SIUL2_Type *base, uint32_t pin, *siul2_port_direction_t* direction)

Set the pin direction.

Parameters

- base – SIUL2 peripheral base pointer
- pin – pin number, 0, 1...511, see RM for available pins.
- direction – pin direction.

void SIUL2_PortSet(SIUL2_Type *base, *siul2_port_num_t* port, uint32_t pins)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – SIUL2 peripheral base pointer
- port – GPIO PORT number, see “siul2_port_num_t”.
- pins – ORed GPIO pins in a port.

void SIUL2_PortMaskWrite(SIUL2_Type *base, *siul2_port_num_t* port, uint32_t pins, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – SIUL2 peripheral base pointer
- port – GPIO PORT number, see “siul2_port_num_t”.
- pins – GPIO pins to be configured.
- mask – ORed bits of which pins will be masked.

void SIUL2_PortClear(SIUL2_Type *base, *siul2_port_num_t* port, uint32_t pins)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – SIUL2 peripheral base pointer
- port – GPIO PORT number, see “siul2_port_num_t”.
- pins – GPIO pin number macro

void SIUL2_PortToggle(SIUL2_Type *base, *siul2_port_num_t* port, uint32_t pins)

Toggle the output level of the multiple GPIO pins.

Parameters

- base – SIUL2 peripheral base pointer
- port – GPIO PORT number, see “siul2_port_num_t”.
- pins – GPIO pin number macro

static inline void SIUL2_PortPinWrite(SIUL2_Type *base, *siul2_port_num_t* port, uint32_t pin, uint8_t output)

Set the pin output.

Parameters

- base – SIUL2 peripheral base pointer

- port – GPIO PORT number, see “siul2_port_num_t”.
- pin – GPIO pin number
- output – Output value, 0 or 1.

static inline uint32_t SIUL2_PortPinRead(SIUL2_Type *base, *siul2_port_num_t* port, uint8_t pin)

Get the pin input.

Parameters

- base – SIUL2 peripheral base pointer
- port – GPIO PORT number, see “siul2_port_num_t”.
- pin – GPIO pin number

static inline void SIUL2_PinWrite(SIUL2_Type *base, uint32_t pin, uint8_t output)

Set the pin output.

Parameters

- base – SIUL2 peripheral base pointer
- pin – GPIO pin number
- output – Output value, 0 or 1.

static inline uint32_t SIUL2_PinRead(SIUL2_Type *base, uint8_t pin)

Get the pin input.

Parameters

- base – SIUL2 peripheral base pointer
- pin – GPIO pin number

static inline void SIUL2_SetGlitchFilterPrescaler(SIUL2_Type *base, uint8_t div)

Set the Glitch filter prescaler.

Parameters

- base – SIUL2 peripheral base pointer
- div – Glitch filter prescaler 0...15. Prescaled Filter Clock period is internal oscillator period x (div + 1).

static inline void SIUL2_SetGlitchFilterMaxCount(SIUL2_Type *base, uint32_t req, int8_t filterCount)

Set Glitch filter max count.

Parameters

- base – SIUL2 peripheral base pointer
- req – which interrupt/DMA request to set, 0...31
- filterCount – Maximum filter count 0...15, < 0 disable filterCount.

void SIUL2_SetDmaInterruptConfig(SIUL2_Type *base, uint32_t req, *siul2_interrupt_config_t* config)

Setup Interupt configuration.

Parameters

- base – SIUL2 peripheral base pointer
- req – which interrupt/DMA request to set, 0...31
- config – the configuration structure.

void SIUL2_EnableExtInterrupt(SIUL2_Type *base, uint32_t req, *siul2_interrupt_config_t* config, int8_t filterCount)

Enable external interrupt.

Parameters

- base – SIUL2 peripheral base pointer
- req – which interrupt request to set, 0...31
- config – interrupt configuration, siul2_interrupt_config_t
- filterCount – Maximum filter count 0...15, < 0 disable filterCount.

void SIUL2_EnableExtDma(SIUL2_Type *base, uint32_t req, *siul2_interrupt_config_t* config, int8_t filterCount)

Enable external DMA request.

Parameters

- base – SIUL2 peripheral base pointer
- req – which DMA request to set, refer to RM for supported external DMA request number.
- config – interrupt configuration, siul2_interrupt_config_t
- filterCount – Maximum filter count 0...15, < 0 disable filterCount.

static inline void SIUL2_DisableExtDmaAndInterrupt(SIUL2_Type *base, uint32_t req)

Disable external DMA and interrupt.

Parameters

- base – SIUL2 peripheral base pointer
- req – which interrupt/DMA request to set, 0...31

static inline void SIUL2_DisableExtDmaAndInterrupts(SIUL2_Type *base, uint32_t mask)

Disable mutliple external DMA and interrupts.

Parameters

- base – SIUL2 peripheral base pointer
- mask – bit mask of external DMA or interupt requests

void SIUL2_EnableExtInterrupts(SIUL2_Type *base, uint32_t mask, *siul2_interrupt_config_t* config, int8_t filterCount)

Enable mutliple external interrupts.

Parameters

- base – SIUL2 peripheral base pointer
- mask – bit mask of interrupt requests
- config – interrupt configuration, siul2_interrupt_config_t
- filterCount – Maximum filter count 0...15, < 0 disable filterCount.

void SIUL2_EnableExtDmaRequests(SIUL2_Type *base, uint32_t mask, *siul2_interrupt_config_t* config, int8_t filterCount)

Enable mutliple external DMA requests.

Parameters

- base – SIUL2 peripheral base pointer
- mask – bit mask of DMA requests

- config – interrupt configuration, siul2_interrupt_config_t
- filterCount – Maximum filter count 0...15, < 0 disable filterCount.

static inline uint32_t SIUL2_GetExtDmaInterruptStatusFlags(SIUL2_Type *base)

Get the external DMA/interrupt status flag.

Parameters

- base – SIUL2 peripheral base pointer

Returns

the status flags

static inline void SIUL2_ClearExtDmaInterruptStatusFlags(SIUL2_Type *base, uint32_t mask)

Clear the external DMA/interrupt status flags.

Parameters

- base – SIUL2 peripheral base pointer
- mask – the status flags bit mask

uint32_t SIUL2_GetPinEirqNumber(uint32_t pin)

Get the external interrupt request number for the pin.

Parameters

- pin – GPIO pin number, 0...511, refer to RM for avaiable Pins.

Returns

0...31 the EIRQ number, > 32 means the pin is not capable for EIRQ.

void SIUL2_EnableExtInteruptWithCallback(SIUL2_Type *base, uint32_t req, *siul2_interrupt_config_t* config, int8_t filterCount, *siul2_cb_t* cb)

Enable EIRQ with callback. This function enables the interrupt for the selected EIRQ. It enables interrupt both in SIUL2 and NIVC. So, the callback function will be called when interrupt comes.

Parameters

- base – SIUL2 peripheral base pointer
- req – which interrupt request to set, 0...31
- config – interrupt configuration, siul2_interrupt_config_t
- filterCount – Maximum filter count 0...15, < 0 disable filterCount.

void SIUL2_DisableExtInterruptCallback(SIUL2_Type *base, uint32_t req)

Disable EIRQ callback.

This function disables the interrupt for the selected EIRQ. It disables interrupt in SIUL2 and NIVC.

Parameters

- base – SIUL2 peripheral base pointer
- req – which interrupt request to set, 0...31

struct siul2_pin_settings

#include <fsl_siul2.h> Defines the converter configuration.

This structure is used to configure the pins

Public Members

SIUL2_Type *base
: The main SIUL2 base pointer.

uint32_t pinPortIdx
: Port pin number.

siul2_port_pull_config_t pullConfig
: Internal resistor pull feature selection.

siul2_port_mux_t mux
: Pin output muxing selection.

siul2_port_safe_mode_t safeMode
: Configures the Safe Mode Control, apply for SIUL2_0/1

siul2_port_slew_rate_t slewRateCtrlSel
: Configures the Slew Rate Control field.

siul2_port_drive_strength_t driveStrength
: Configures DSE

siul2_port_input_filter_t inputFilter
: Configures IFE

siul2_port_pull_keep_t pullKeep
: Configures PKE

siul2_port_invert_t invert
: Configures IFE

siul2_port_output_buffer_t outputBuffer
: Configures the Output Buffer Enable.

siul2_port_input_buffer_t inputBuffer
: Configures the Input Buffer Enable.

siul2_adc_interleaves_t adcInterleaves[(2U)]
: Configures the adc interleave mux modes.

siul2_port_inputmux_t inputMux[(16U)]
: Configures the input muxing

uint32_t inputMuxReg[(16U)]
: Configures the input muxing register. For the pins controlled by both SIUL2_0 and SIUL2_1 instances, refer the note for PINS_DRV_SetInputBuffer function

siul2_port_pins_Level_t initValue
: Initial value

struct siul2_filter_config
: *#include <fsl_siul2.h>* SIUL2 Interrupt Filter Configuration.

This structure is used to configure the filter for interrupt.

Public Members

uint8_t preScaler
: Interrupt Filter clock prescaler setting, 0-15.

uint8_t maxCount
: Interrupt Filter Maximum Counter, 0-15.

2.61 STM: STM Driver

void STM_GetDefaultConfig(*stm_config_t* *config)

Initializes STM configure structure.

This function initializes the STM configure structure to default value. The default value are:

```
config->enableRunInDebug = true;
config->enableIRQ = true;
config->prescale = 0U;
```

See also:

stm_config_t

Parameters

- config – Pointer to STM config structure.

void STM_Init(STM_Type *base, const *stm_config_t* *config)

Initializes the STM.

This function configures the peripheral for basic operation.

Parameters

- base – STM peripheral base address
- config – The configuration of STM

void STM_Deinit(STM_Type *base)

Shuts down the STM.

This function shuts down the STM.

Parameters

- base – STM peripheral base address

static inline void STM_ClearStatusFlags(STM_Type *base, *stm_channel_t* channel)

Clear STM flag.

This function clears STM channel interrupt flag which is set due to a match on the channel.

See also:

stm_channel_t

Parameters

- base – STM peripheral base address
- channel – The compare channel that corresponds to the flag that needs to be cleared.

static inline uint32_t STM_GetStatusFlags(STM_Type *base, *stm_channel_t* channel)

Gets channel interrupt flag of the STM.

See also:

stm_channel_t

Parameters

- base – STM peripheral base address
- channel – The channel number.

Returns
The channel Interrupt flag.

void STM_SetCompare(STM_Type *base, *stm_channel_t* channel, uint32_t value)
Set compare value for specific channel of the STM module and enable the channel.

See also:
stm_channel_t

Parameters

- base – STM peripheral base address
- channel – The compare channel to be configured.
- value – Compare value, range from 0 to 0xFFFFFFFF

static inline void STM_EnableCompareChannel(STM_Type *base, *stm_channel_t* channel)
Enable STM compare channel.

See also:
stm_channel_t

Parameters

- base – STM peripheral base address
- channel – The channel number.

static inline void STM_DisableCompareChannel(STM_Type *base, *stm_channel_t* channel)
Disable STM compare channel.

See also:
stm_channel_t

Parameters

- base – STM peripheral base address
- channel – The channel number.

FSL_STM_DRIVER_VERSION
Defines STM driver version.

enum _stm_channel
List of STM channels.

Values:

enumerator kSTM_Channel_0
STM channel 0

enumerator kSTM_Channel_1
STM channel 1

enumerator kSTM_Channel_2
STM channel 2

enumerator kSTM_Channel_3
STM channel 3

enum _stm_channel_match
List of STM compare match mask to indicate channel interrupt in stm_callback_t function flags parameters.

Values:

enumerator kSTM_Channel_Match_Msk_0
STM channel compare register 0

enumerator kSTM_Channel_Match_Msk_1
STM channel compare register 1

enumerator kSTM_Channel_Match_Msk_2
STM channel compare register 2

enumerator kSTM_Channel_Match_Msk_3
STM channel compare register 3

typedef enum *_stm_channel* stm_channel_t
List of STM channels.

typedef enum *_stm_channel_match* stm_channel_match_t
List of STM compare match mask to indicate channel interrupt in stm_callback_t function flags parameters.

typedef struct *_stm_config* stm_config_t
Describes STM configuration structure.

typedef void (*stm_callback_t)(uint32_t flags)
Define STM interrupt callback function pointer.

void STM_DriverIRQHandler(uint32_t index)

static inline void STM_StartTimer(STM_Type *base)
Start the STM module.

This function write value into STM_CR register to enable the STM

Parameters

- base – STM peripheral base address

static inline void STM_StopTimer(STM_Type *base)
Stop the STM module.

This function write value into STM_CR register to disable the STM.

Parameters

- base – STM peripheral base address

struct _stm_config
#include <fsl_stm.h> Describes STM configuration structure.

Public Members

bool enableRunInDebug
true: Timer stops in Debug mode false: Timer runs in Debug mode

bool enableIRQ

true: Enable STM interrupt false: Disable STM interrupt

uint8_t prescale

Selects the module clock divide value for the prescale (0–255).

2.62 SWT: Software Watchdog Timer

void SWT_Init(SWT_Type *base, const *swt_config_t* *config)

Initializes the SWT module with configuration.

This function initializes the SWT. When called, the SWT runs according to the configuration. This is an example.

```
swt_config_t config;
SWT_GetDefaultConfig(&config);
config.timeoutValue = 32000U;
SWT_Init(swt_base, &config);
```

Parameters

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

void SWT_Deinit(SWT_Type *base)

De-initializes the SWT module.

This function de-initializes the SWT.

Parameters

- base – SWT peripheral base address

void SWT_GetDefaultConfig(*swt_config_t* *config)

Gets the default configuration for SWT.

This function initializes the SWT configuration structure to a default value. The default values are as follows.

```
config->enableSwt = true;
config->enableRunInDebug = false;
config->enableRunInStop = true;
config->interruptThenReset = false;
config->enableWindowMode = false;
config->resetOnInvalidAccess = true;
config->serviceMode = kSWT_FixedServiceSequence;
config->timeoutValue = 0xFFFFU;
config->windowValue = 0U;
config->serviceKey = 0U;
```

Parameters

- config – Pointer to the configuration structure.

static inline void SWT_Enable(SWT_Type *base)

Enable SWT module.

Parameters

- base – SWT peripheral base address

static inline void SWT_Disable(SWT_Type *base)

Disable SWT module.

Parameters

- base – SWT peripheral base address

static inline void SWT_HardLock(SWT_Type *base)

Enable the hard lock.

This function will hard lock the SWT registers. Hard lock is disabled only by a reset.

Parameters

- base – SWT peripheral base address

static inline void SWT_SoftLock(SWT_Type *base)

Enable the soft lock.

This function will soft lock the SWT registers. Soft lock is disabled by a reset or by writing the unlock sequence.

Parameters

- base – SWT peripheral base address

void SWT_SoftUnlock(SWT_Type *base)

Unlock the soft lock.

This function will unlock the SWT registers locked by SWT_SoftLock.

Parameters

- base – SWT peripheral base address

void SWT_RefreshWithFixedServiceSequence(SWT_Type *base)

SWT Refresh with Fixed Service Sequence.

This function will execute a fixed service sequence to refresh the SWT. SWT shall work in fixed service sequence mode swt_service_mode_t.

Parameters

- base – SWT peripheral base address

static inline void SWT_SetServiceKey(SWT_Type *base, uint16_t serviceKey)

Set the service key.

This function will set the initial service key for the keyed service sequence.

Parameters

- base – SWT peripheral base address
- serviceKey – The service key

void SWT_RefreshWithKeyedServiceSequence(SWT_Type *base)

SWT Refresh with Keyed Service Sequence.

This function will excute a keyed service sequence to refresh the SWT. SWT shall work in keyed service sequence mode swt_service_mode_t.

Parameters

- base – SWT peripheral base address

void SWT_Refresh(SWT_Type *base)

SWT Refresh with automatic service sequence.

This function will automatically select the service sequence to refresh the SWT.

Parameters

- base – SWT peripheral base address

void SWT_SetTimeoutValue(SWT_Type *base, uint32_t timeoutValue)

Set the timeout value.

This function will set the SWT timeout period in clock cycles.

Parameters

- base – SWT peripheral base address
- timeoutValue – The timeout value

static inline void SWT_SetWindowValue(SWT_Type *base, uint32_t windowValue)

Set the window start value.

This function will set the window start value for SWT window mode. Software can write the service sequence only when the internal timer is less than this value.

Parameters

- base – SWT peripheral base address
- windowValue – The window start value

static inline uint32_t SWT_GetCounterValue(SWT_Type *base)

Get the counter value.

This function will get the internal counter value when SWT is disabled. It is usually used to determine whether the internal countdown timer is working properly.

This is an example:

```
SWT_Enable(SWT_BASE);
Delay(the_delay_time_shall_less_than_timeout_value);
SWT_Disable(SWT_BASE);
uint32_t counter_value = SWT_GetCounterValue(SWT_BASE);
```

Parameters

- base – SWT peripheral base address

Returns

The counter value

static inline bool SWT_GetTimeoutInterruptFlag(SWT_Type *base)

Get the timeout interrupt flag.

This function will get the timeout interrupt flag.

Parameters

- base – SWT peripheral base address

Returns

The timeout interrupt flag. true: Interrupt request due to an initial timeout false: No interrupt request

static inline void SWT_ClearTimeoutInterruptFlag(SWT_Type *base)

Clear the timeout interrupt flag.

This function will clear the timeout interrupt flag and then SWT will send interrupt request due to an initial timeout.

Parameters

- base – SWT peripheral base address

static inline bool SWT_GetTimeoutResetFlag(SWT_Type *base)

Get the timeout reset flag.

This funcion can get the SWT reset request flag.

Parameters

- base – SWT peripheral base address

Returns

The reset request flag. true: Any reset request initiated false: No reset request

static inline void SWT_ClearTimeoutResetFlag(SWT_Type *base)

Clear the timeout reset flag.

This funcion can only reset the SWT module instead of the whole system. See the chip-specific information for the specific event associated with this flag.

Parameters

- base – SWT peripheral base address

FSL_SWT_DRIVER_VERSION

SWT Driver Version 2.1.0.

SWT_FIRST_WORD_OF_SOFT_UNLOCK

First word of soft unlock sequence

SWT_SECOND_WORD_OF_SOFT_UNLOCK

Second word of soft unlock sequence

SWT_FIRST_WORD_OF_FIXED_SERVICE

First word of fixed service sequence

SWT_SECOND_WORD_OF_FIXED_SERVICE

Second word of fixed service sequence

enum _swt_service_mode_t

SWT service mode.

Values:

enumerator kSWT_FixedServiceSequence

Write the fixed sequence

enumerator kSWT_KeyedServiceSequence

write two pseudorandom key values

typedef enum *_swt_service_mode_t* swt_service_mode_t

SWT service mode.

typedef struct *_swt_config* swt_config_t

SWT configuration structure.

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

struct _swt_config

#include <fsl_swt.h> SWT configuration structure.

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

Public Members

bool enableSwt
: Enables SWT

bool enableRunInDebug
: Enable timers continues in debug mode

bool enableRunInStop
: Enable timers continues in stop/standby mode

bool interruptThenReset
: true: Generate an interrupt and load the countdown timer on an inital timeout, then generate a reset request on a subsequent timeout. false: Generate an immediate reset request on any timeout.

bool enableWindowMode
: Enable timers run in window mode

bool resetOnInvalidAccess
: true: Generate a bus error and reset request; false: Generate a bus error

swt_service_mode_t serviceMode
: Service mode, swt_service_mode_t

uint32_t timeoutValue
: Watchdog timeout period in clock cycles

uint32_t windowValue
: Window start value

uint16_t serviceKey
: Service key

2.63 TEMPSENSE: Temperature Sensor Module

void TEMPSENSE_GetDefaultConfig(*tempsense_config_t* *config)
: This function is used to get predefined configurations for the tempsense initialization.

  Parameters

  - config – Pointer to the tempsense configuration structure, please refer to tempsense_config_t for details.

void TEMPSENSE_Init(TEMPSENSE_Type *base, const *tempsense_config_t* *config)
: This function is used to initialize the tempsense.

  Parameters

  - base – Tempsense peripheral base address.
  - config – Pointer to the tempsense configuration structure, please refer to tempsense_config_t for details.

void TEMPSENSE_Deinit(TEMPSENSE_Type *base)
: This function is used to de-initialize the tempsense.

  Parameters

  - base – Tempsense peripheral base address.

float TEMPSENSE_GetCurrentTemperature(TEMPSENSE_Type *base, uint16_t adcResult, float adcRef, uint8_t adcRes)

Get current temperature.

Parameters

- base – Tempsense base pointer
- adcResult – The ADC conversion result.
- adcRef – The ADC VREF which used to calculate the real temperature.
- adcRes – The ADC resolution.

Returns

current temperature with degrees Celsius.

static inline void TEMPSENSE_EnableTempsense(TEMPSENSE_Type *base, bool enable)

Enable tempsense.

Parameters

- base – Tempsense base pointer.
- enable – Indicates whether to enable the tempsense.
  - **true** Enable the tempsense.
  - **false** Disable the tempsense.

static inline void TEMPSENSE_ExposeGround(TEMPSENSE_Type *base, bool enable)

Expose ground.

Parameters

- base – TEMPMON base pointer.
- enable – Indicates whether to expose the ground.
  - **true** Expose the ground.
  - **false** No exposure of the ground.

FSL_TEMPSENSE_DRIVER_VERSION

Tempsense driver version 2.0.0.

typedef struct *_tempsense_config* tempsense_config_t

This structure is used to configure the tempsense.

struct _tempsense_config

#include <fsl_tempsense.h> This structure is used to configure the tempsense.

Public Members

bool exposeGround

Decides whether to expose ground on the ADC output.

2.64 TRGMUX: Trigger Mux Driver

static inline void TRGMUX_LockRegister(TRGMUX_Type *base, uint32_t index)

Sets the flag of the register which is used to mark writeable.

The function sets the flag of the register which is used to mark writeable. Example:

```
TRGMUX_LockRegister(TRGMUX0,kTRGMUX_Trgmux0Dmamux0);
```

Parameters

- base – TRGMUX peripheral base address.
- index – The index of the TRGMUX register, see the enum trgmux_device_t defined in <SOC>.h.

status_t TRGMUX_SetTriggerSource(TRGMUX_Type *base, uint32_t index, *trgmux_trigger_input_t* input, uint32_t trigger_src)

Configures the trigger source of the appointed peripheral.

The function configures the trigger source of the appointed peripheral. Example:

```
TRGMUX_SetTriggerSource(TRGMUX0, kTRGMUX_Trgmux0Dmamux0, kTRGMUX_TriggerInput0,
↪ kTRGMUX_SourcePortPin);
```

Parameters

- base – TRGMUX peripheral base address.
- index – The index of the TRGMUX register, see the enum trgmux_device_t defined in <SOC>.h.
- input – The MUX select for peripheral trigger input
- trigger_src – The trigger inputs for various peripherals. See the enum trgmux_source_t defined in <SOC>.h.

Return values

- kStatus_Success – Configured successfully.
- kStatus_TRGMUX_Locked – Configuration failed because the register is locked.

FSL_TRGMUX_DRIVER_VERSION

TRGMUX driver version.

TRGMUX configure status.

Values:

enumerator kStatus_TRGMUX_Locked

Configure failed for register is locked

enum _trgmux_trigger_input

Defines the MUX select for peripheral trigger input.

Values:

enumerator kTRGMUX_TriggerInput0

The MUX select for peripheral trigger input 0

enumerator kTRGMUX_TriggerInput1

The MUX select for peripheral trigger input 1

enumerator kTRGMUX_TriggerInput2

The MUX select for peripheral trigger input 2

enumerator kTRGMUX_TriggerInput3

The MUX select for peripheral trigger input 3

typedef enum *_trgmux_trigger_input* trgmux_trigger_input_t

Defines the MUX select for peripheral trigger input.

2.65 TSPC: Touch Sensing Pin Coupling

void TSPC_Init(TSPC_Type *base)

Initializes the TSPC peripheral.

This function ungates the TSPC clock.

Parameters

- base – TSPC peripheral base address.

void TSPC_Deinit(TSPC_Type *base)

Deinitializes the TSPC peripheral.

This function gates the TSPC clock.

Parameters

- base – TSPC peripheral base address.

void TSPC_InitGroup(TSPC_Type *base, *tspc_group_t* port, uint64_t padsMask)

Initializes the TSPC group.

This function configures which pads will participate in the grouping and enables grouping.

Parameters

- base – XBIC peripheral base address.
- group – number of TSPC group.
- padsMask – Mask value of TSPC pads, tspc_group1_pads_t or tspc_group2_pads_t.

static inline void TSPC_EnableGroup(TSPC_Type *base, *tspc_group_t* group)

Enable TSPC group.

Parameters

- base – TSPC peripheral base address.
- group – number of TSPC group.

static inline void TSPC_DisableGroup(TSPC_Type *base, *tspc_group_t* group)

Disable TSPC group.

Parameters

- base – TSPC peripheral base address.
- group – The group number of TSPC.

FSL_TSPC_DRIVER_VERSION

TSPC driver version 2.0.0.

enum _tspc_group

_tspc_group TSPC group.

Values:

enumerator kTSPC_Group1

TSPC group 0.

enumerator kTSPC_Group2
TSPC group 1.

typedef enum *_tspc_group* tspc_group_t
_tspc_group TSPC group.

2.66 VIRT_WRAPPER: Virtualization Wrapper

FSL_VIRT_WRAPPER_DRIVER_VERSION
Version.

status_t VIRT_WRAPPER_MapPins(VIRT_WRAPPER_Type *base, uint16_t port, uint32_t pins, virt_wrapper_slot_t slot)
Maps the pins to a specific slot.

This function maps the pins of a port to specific slot. This function can map multiple pins at one time, by using a bit mask.

Parameters

- base – VIRT_WRAPPER peripheral base address.
- port – Pin port number, starting from 0, for exapmle, PTA is 0, PTB is 1.
- pins – Bit-mask of the pins, ((1 « 0) | (1 « 1)) means pin 0 and pin 1 in the port.
- slot – The slot to map to.

Returns
kStatus_Success on success, otherwise returns error code.

status_t VIRT_WRAPPER_MapInputMux(VIRT_WRAPPER_Type *base, uint32_t inputMuxIndex, virt_wrapper_slot_t slot)
Maps the input multiplexer to a specific slot.

This function maps the input multiplexer to a specified slot.

Parameters

- base – VIRT_WRAPPER peripheral base address.
- inputMuxIndex – Index of the input multiplexer.
- slot – The slot to map to.

Returns
kStatus_Success on success, otherwise returns error code.

static inline void VIRT_WRAPPER_MapInterrupt(VIRT_WRAPPER_Type *base, virt_wrapper_slot_t slot)
Maps interrupt control to a specific slot.

This function maps the interrupt control register to a specified slot.

Parameters

- base – VIRT_WRAPPER peripheral base address.
- slot – The slot to map to.

static inline void VIRT_WRAPPER_MapGCR(VIRT_WRAPPER_Type *base, virt_wrapper_slot_t slot)
Maps the GPC register to a specific slot.

This function maps the GPC register to a specified slot.

Parameters

- base – VIRT_WRAPPER peripheral base address.
- slot – The slot to map to.

2.67 WKPU: Wakeup Unit driver

void WKPU_GetDefaultExternalWakeUpSourceConfig(*wkpu_external_wakeup_source_config_t* *config)

Fills in the WKPU external wake up source config struct with the default settings.

The default values are as follows.

```
config->event              = kWKPU_WakeUp;
config->edge                 = kWKPU_PinRisingEdge;
config->enableFilter           = false;
```

Parameters

- config – Pointer to the user defined WKPU configuration structure.

void WKPU_SetExternalWakeUpSourceConfig(WKPU_Type *base, uint64_t mask, const *wkpu_external_wakeup_source_config_t* *config)

Enable and config the external wakeup source.

This function enables/disables the external wake up source as the wake up input. The wake up sources are mostly wake up pins with several internal wakeup modules.

Parameters

- base – WKPU peripheral base address.
- mask – The wake up source to used. This is a logical OR of members of the enumeration wkpu_source_t
- config – Pointer to wkpu_external_wakeup_source_config_t structure.

void WKPU_ClearExternalWakeupSourceConfig(WKPU_Type *base, uint64_t mask)

Disable and clear external wakeup source settings.

Parameters

- base – WKPU peripheral base address.
- mask – The wake up source to used. This is a logical OR of members of the enumeration wkpu_source_t

static inline uint64_t WKPU_GetExternalWakeUpSourceFlag(WKPU_Type *base)

Get the external wakeup source flag.

This function return the external wakeup source flag of the source index.

Parameters

- base – WKPU peripheral base address.

Returns

Wakeup flag of the source index.

static inline void WKPU_ClearExternalWakeUpSourceFlag(WKPU_Type *base, uint64_t mask)

Clear the external wake up source flag.

This function clears external wakeup source flag of the source index .

Parameters

- base – WKPU peripheral base address.
- mask – The wake up source to used. This is a logical OR of members of the enumeration wkpu_source_t

void WKPU_GetDefaultNMIWakeUpConfig(*wkpu_nmi_wakeup_config_t* *config)

Fills in the NMI wake up source config struct with the default settings.

The default values are as follows.

```
config->edge                = kWKPU_PinRisingEdge;
config->enableFilter        = true;
config->enableWakeup        = true;
config->lockRegister        = false;
```

Parameters

- config – Pointer to the user defined WKPU configuration structure.

void WKPU_SetNMIWakeupConfig(WKPU_Type *base, const *wkpu_nmi_wakeup_config_t* *config)

Config NMI event as the wake up soures.

This function configs the NMI as wake up source.

Parameters

- base – WKPU peripheral base address.
- config – Pointer to wkpu_external_wakeup_source_config_t structure.

void WKPU_ClearNMIWakeupConfig(WKPU_Type *base)

Disable and clear NMI settings.

Parameters

- base – WKPU peripheral base address.

static inline uint32_t WKPU_GetNMIStatusFlag(WKPU_Type *base)

static inline void WKPU_ClearNMIStatusFlag(WKPU_Type *base, uint32_t mask)

Clear the NMI status flag.

Parameters

- base – WKPU peripheral base address.
- mask – The NMI status flag to be cleared. This is a logical OR of members of the enumeration wkpu_nmi_status_flag_t

FSL_WKPU_DRIVER_VERSION

Defines WKPU driver version 2.0.0.

enum _wkpu_source

The wakeup source enumeration.

Values:

enumerator kWKPU_SourceNone

No wakeup source

enumerator kWKPU_Source0

SWT_0 timeout and RTC-API API wakeup source

enumerator kWKPU_Source1

RTC-API RTC timeout wakeup source

enumerator kWKPU_Source2
Round robin wakeup source from LPCMP_0, LPCMP_1, or LPCMP_2

enumerator kWKPU_Source3
PIT0-RTI wakeup source

enumerator kWKPU_Source4
External pin wakeup source WKPU[0]

enumerator kWKPU_Source5
External pin wakeup source WKPU[1]

enumerator kWKPU_Source6
External pin wakeup source WKPU[2]

enumerator kWKPU_Source7
External pin wakeup source WKPU[3]

enumerator kWKPU_Source8
External pin wakeup source WKPU[4]

enumerator kWKPU_Source9
External pin wakeup source WKPU[5]

enumerator kWKPU_Source10
External pin wakeup source WKPU[6]

enumerator kWKPU_Source11
External pin wakeup source WKPU[7]

enumerator kWKPU_Source12
External pin wakeup source WKPU[8]

enumerator kWKPU_Source13
External pin wakeup source WKPU[9]

enumerator kWKPU_Source14
External pin wakeup source WKPU[10]

enumerator kWKPU_Source15
External pin wakeup source WKPU[11]

enumerator kWKPU_Source16
External pin wakeup source WKPU[12]

enumerator kWKPU_Source17
External pin wakeup source WKPU[13]

enumerator kWKPU_Source18
External pin wakeup source WKPU[14]

enumerator kWKPU_Source19
External pin wakeup source WKPU[15]

enumerator kWKPU_Source20
External pin wakeup source WKPU[16]

enumerator kWKPU_Source21
External pin wakeup source WKPU[17]

enumerator kWKPU_Source22
External pin wakeup source WKPU[18]

enumerator kWKPU_Source23
External pin wakeup source WKPU[19]

enumerator kWKPU_Source24
External pin wakeup source WKPU[20]

enumerator kWKPU_Source25
External pin wakeup source WKPU[21]

enumerator kWKPU_Source26
External pin wakeup source WKPU[22]

enumerator kWKPU_Source27
External pin wakeup source WKPU[23]

enumerator kWKPU_Source28
External pin wakeup source WKPU[24]

enumerator kWKPU_Source29
External pin wakeup source WKPU[25]

enumerator kWKPU_Source30
External pin wakeup source WKPU[26]

enumerator kWKPU_Source31
External pin wakeup source WKPU[27]

enumerator kWKPU_Source32
External pin wakeup source WKPU[28]

enumerator kWKPU_Source33
External pin wakeup source WKPU[29]

enumerator kWKPU_Source34
External pin wakeup source WKPU[30]

enumerator kWKPU_Source35
External pin wakeup source WKPU[31]

enumerator kWKPU_Source36
External pin wakeup source WKPU[32]

enumerator kWKPU_Source37
External pin wakeup source WKPU[33]

enumerator kWKPU_Source38
External pin wakeup source WKPU[34]

enumerator kWKPU_Source39
External pin wakeup source WKPU[35]

enumerator kWKPU_Source40
External pin wakeup source WKPU[36]

enumerator kWKPU_Source41
External pin wakeup source WKPU[37]

enumerator kWKPU_Source42
External pin wakeup source WKPU[38]

enumerator kWKPU_Source43
External pin wakeup source WKPU[39]

enumerator kWKPU_Source44
External pin wakeup source WKPU[40]

enumerator kWKPU_Source45
External pin wakeup source WKPU[41]

enumerator kWKPU_Source46
External pin wakeup source WKPU[42]

enumerator kWKPU_Source47
External pin wakeup source WKPU[43]

enumerator kWKPU_Source48
External pin wakeup source WKPU[44]

enumerator kWKPU_Source49
External pin wakeup source WKPU[45]

enumerator kWKPU_Source50
External pin wakeup source WKPU[46]

enumerator kWKPU_Source51
External pin wakeup source WKPU[47]

enumerator kWKPU_Source52
External pin wakeup source WKPU[48]

enumerator kWKPU_Source53
External pin wakeup source WKPU[49]

enumerator kWKPU_Source54
External pin wakeup source WKPU[50]

enumerator kWKPU_Source55
External pin wakeup source WKPU[51]

enumerator kWKPU_Source56
External pin wakeup source WKPU[52]

enumerator kWKPU_Source57
External pin wakeup source WKPU[53]

enumerator kWKPU_Source58
External pin wakeup source WKPU[54]

enumerator kWKPU_Source59
External pin wakeup source WKPU[55]

enumerator kWKPU_Source60
External pin wakeup source WKPU[56]

enumerator kWKPU_Source61
External pin wakeup source WKPU[57]

enumerator kWKPU_Source62
External pin wakeup source WKPU[58]

enumerator kWKPU_Source63
External pin wakeup source WKPU[59]

enumerator kWKPU_SourceAll
All wakeup sources

enum _wkpu_pin_edge_detection

Wake up pin edge detection enumeration, applied for both NMI and external wake up and interrupt pin.

Values:

enumerator kWKPU_PinDisable

Pin disabled.

enumerator kWKPU_PinRisingEdge

Pin enabled with the rising edge detection.

enumerator kWKPU_PinFallingEdge

Pin enabled with the falling edge detection.

enumerator kWKPU_PinAnyEdge

Pin enabled with any update detection.

enum _wkpu_event

wake up event enumeration.

Values:

enumerator kWKPU_Interrupt

Interrupt.

enumerator kWKPU_WakeUp

Wake up

enumerator kWKPU_InterruptAndWakeUp

Interrupt and wake up

enum _wkpu_nmi_status_flag

NMI status flag enumeration.

Values:

enumerator kWKPU_NMIOverrunStatusFlag

NMI Overrun Status Flag.

enumerator kWKPU_NMIStatusFlag

NMI status flag. Assert when NMI rising-edge or falling-edge event occurred.

enumerator kWKPU_NMIAllStatusFlag

All NMI status flag.

typedef enum *_wkpu_source* wkpu_source_t

The wakeup source enumeration.

typedef enum *_wkpu_pin_edge_detection* wkpu_pin_edge_detection_t

Wake up pin edge detection enumeration, applied for both NMI and external wake up and interrupt pin.

typedef enum *_wkpu_event* wakeup_event_t

wake up event enumeration.

typedef enum *_wkpu_nmi_status_flag* wkpu_nmi_status_flag_t

NMI status flag enumeration.

typedef struct *_wkpu_external_wakeup_source_config* wkpu_external_wakeup_source_config_t

External WakeUp pin configuration.

typedef struct *_wkpu_nmi_wakeup_config* wkpu_nmi_wakeup_config_t

NMI wake up configuration.

typedef void (*wkpu_callback_t)(uint64_t mask)
WakeUp callback function.

void WKPU_RegisterCallBack(*wkpu_callback_t* cb_func)
Register callback.

Parameters

- cb_func – callback function

struct _wkpu_external_wakeup_source_config
#include <fsl_wkpu.h> External WakeUp pin configuration.

Public Members

wakeup_event_t event
External Input wakeup Pin event

wkpu_pin_edge_detection_t edge
External Input pin edge detection.

bool enableFilter
Enable filter for the external input pin.

struct _wkpu_nmi_wakeup_config
#include <fsl_wkpu.h> NMI wake up configuration.

Public Members

wkpu_pin_edge_detection_t edge
External Input pin edge detection.

bool enableFilter
Enable filter for the NMI input pin.

bool enableWakeup
Enable wakeup for the NMI input pin.

bool lockRegister
Enable NMI for the NMI input pin.

2.68 XBIC: Crossbar Integrity Checker

void XBIC_EnableMasterPortEDC(XBIC_Type *base, xbic_master_port_t masterPort)
Enables individual XBIC master port error detection.

This function enables individual XBIC master port error detection.

Parameters

- base – XBIC peripheral base address.
- masterPort – ID of XBIC master port.

void XBIC_EnableSlavePortEDC(XBIC_Type *base, xbic_slave_port_t slavePort)
Enables individual XBIC slave port error detection.

This function enables individual XBIC slave port error detection.

Parameters

- base – XBIC peripheral base address.
- slavePort – ID of XBIC slave port.

void XBIC_DisableMasterPortEDC(XBIC_Type *base, xbic_master_port_t masterPort)

Disables individual XBIC master port error detection.

This function disables individual XBIC master port error detection.

Parameters

- base – XBIC peripheral base address.
- masterPort – ID of XBIC master port.

void XBIC_DisableSlavePortEDC(XBIC_Type *base, xbic_slave_port_t slavePort)

Disables individual XBIC slave port error detection.

This function disables individual XBIC slave port error detection.

Parameters

- base – XBIC peripheral base address.
- slavePort – ID of XBIC slave port.

void XBIC_ErrorInjectionConfig(XBIC_Type *base, xbic_master_port_t masterPort, xbic_slave_port_t slavePort, *xbic_error_syndromes_t* syndromes)

Sets error injection parameters config for selected XBIC master and slave port.

This function sets error injection parameters for selected XBIC master and slave port.

Parameters

- base – XBIC peripheral base address.
- masterPort – ID of XBIC master port.
- slavePort – ID of XBIC slave port.
- syndromes – value of XBIC error syndromes.

static inline void XBIC_EnableErrorInjection(XBIC_Type *base, bool enable)

Enable XBIC error injection.

This function enables or disable XBIC error injection. This API should be called after calling XBIC_ErrorInjectionConfig.

Parameters

- base – XBIC peripheral base address.
- enable – Switcher of XBIC error injection feature. “true” means to enable, “false” means not.

static inline uint32_t XBIC_GetErrorValidFlag(XBIC_Type *base)

Get XBIC error status valid flag. This function gets XBIC error status valid flag.

Parameters

- base – XBIC peripheral base address.

Returns

error valid status. (0-no error detected, 1-error detected)

static inline uint32_t XBIC_GetDpseFlag(XBIC_Type *base, xbic_slave_port_t slavePort)

Get selected master port feedback integrity error status.

Parameters

- base – XBIC peripheral base address.

- slavePort – ID of XBIC slave port.

Returns
Slave port error status (0-no error detected, 1-error detected)

static inline uint32_t XBIC_GetDpmeFlag(XBIC_Type *base, xbic_master_port_t masterPort)

Returns selected slave port feedback integrity error status.

Parameters

- base – XBIC peripheral base address.
- masterPort – ID of XBIC master port.

Returns
Master port error status (0-no error detected, 1-error detected)

static inline uint32_t XBIC_GetErrorSlavePorts(XBIC_Type *base)

Get ID of a slave port targeted by the most recent transfer.

Parameters

- base – XBIC peripheral base address.

Returns
Slave port error status mask xbic_slave_port_t.

static inline uint32_t XBIC_GetErrorMasterPorts(XBIC_Type *base)

Get ID of a master port that requested the most recent transfer.

Parameters

- base – XBIC peripheral base address.

Returns
Master port error status mask xbic_master_port_t.

static inline uint32_t XBIC_GetErrorSyndrome(XBIC_Type *base)

Get the syndrome calculated for the most recent transfer.

Parameters

- base – XBIC peripheral base address.

Returns
Syndrome value.

static inline uint32_t XBIC_GetErrorAddress(XBIC_Type *base)

Get the address of the most recent transfer with an attribute integrity check error detected.

Parameters

- base – XBIC peripheral base address.

Returns
Error address value.

FSL_XBIC_DRIVER_VERSION

XBIC driver version 2.0.1.

enum _xbic_error_syndromes

_xbic_error_syndromes XBIC error syndromes.

Values:

enumerator kXBIC_SynHwrite

hwrite signal.

enumerator kXBIC_SynHtrans0
htrans[0] signal

enumerator kXBIC_SynHsize2
hsize[2] signal

enumerator kXBIC_SynHsize1
hsize[1] signal

enumerator kXBIC_SynHsize0
hsize[0] signal

enumerator kXBIC_SynHprot5
hprot[5] signal

enumerator kXBIC_SynHprot4
hprot[4] signal

enumerator kXBIC_SynHprot3
hprot[3] signal

enumerator kXBIC_SynHprot2
hprot[2] signal

enumerator kXBIC_SynHprot1
hprot[1] signal

enumerator kXBIC_SynHprot0
hprot[0] signal

enumerator kXBIC_SynHburst2
hburst[2] signal

enumerator kXBIC_SynHburst1
hburst[1] signal

enumerator kXBIC_SynHburst0
hburst[0] signal

enumerator kXBIC_SynHmastlock
hmastlock signal

enumerator kXBIC_SynHunalign
hunalign signal

enumerator kXBIC_SynEdc7
edc[7] signal

enumerator kXBIC_SynEdc6
edc[6] signal

enumerator kXBIC_SynEdc5
edc[5] signal

enumerator kXBIC_SynEdc4
edc[4] signal

enumerator kXBIC_SynEdc3
edc[3] signal

enumerator kXBIC_SynEdc2
edc[2] signal

enumerator kXBIC_SynEdc1
edc[1] signal

enumerator kXBIC_SynEdc0
edc[0] signal

enumerator kXBIC_SynHbstrb7
hbstrb[7] signal

enumerator kXBIC_SynHbstrb6
hbstrb[6] signal

enumerator kXBIC_SynHbstrb5
hbstrb[5] signal

enumerator kXBIC_SynHbstrb4
hbstrb[4] signal

enumerator kXBIC_SynHbstrb3
hbstrb[3] signal

enumerator kXBIC_SynHbstrb2
hbstrb[2] signal

enumerator kXBIC_SynHbstrb1
hbstrb[1] signal

enumerator kXBIC_SynHbstrb0
hbstrb[0] signal

enumerator kXBIC_SynHmaster3
hmaster[3] signal

enumerator kXBIC_SynHmaster2
hmaster[2] signal

enumerator kXBIC_SynHmaster1
hmaster[1] signal

enumerator kXBIC_SynHmaster0
hmaster[0] signal

enumerator kXBIC_SynHslave2
hslave[2] signal

enumerator kXBIC_SynHslave1
hslave[1] signal

enumerator kXBIC_SynHslave0
hslave[0] signal

enumerator kXBIC_SynHdecorated
hdecorated signal

enumerator kXBIC_SynHdecor31
hdecor[31] signal

enumerator kXBIC_SynHdecor30
hdecor[30] signal

enumerator kXBIC_SynHdecor29
hdecor[29] signal

enumerator kXBIC_SynHdecor28
hdecor[28] signal

enumerator kXBIC_SynHdecor27
hdecor[27] signal

enumerator kXBIC_SynHdecor26
hdecor[26] signal

enumerator kXBIC_SynHdecor25
hdecor[25] signal

enumerator kXBIC_SynHdecor24
hdecor[24] signal

enumerator kXBIC_SynHdecor23
hdecor[23] signal

enumerator kXBIC_SynHdecor22
hdecor[22] signal

enumerator kXBIC_SynHdecor21
hdecor[21] signal

enumerator kXBIC_SynHdecor20
hdecor[20] signal

enumerator kXBIC_SynHdecor19
hdecor[19] signal

enumerator kXBIC_SynHdecor18
hdecor[18] signal

enumerator kXBIC_SynHdecor17
hdecor[17] signal

enumerator kXBIC_SynHdecor16
hdecor[16] signal

enumerator kXBIC_SynHdecor15
hdecor[15] signal

enumerator kXBIC_SynHdecor14
hdecor[14] signal

enumerator kXBIC_SynHdecor13
hdecor[13] signal

enumerator kXBIC_SynHdecor12
hdecor[12] signal

enumerator kXBIC_SynHdecor11
hdecor[11] signal

enumerator kXBIC_SynHdecor10
hdecor[10] signal

enumerator kXBIC_SynHdecor9
hdecor[9] signal

enumerator kXBIC_SynHdecor8
hdecor[8] signal

enumerator kXBIC_SynHdecor7
hdecor[7] signal

enumerator kXBIC_SynHdecor6
hdecor[6] signal

enumerator kXBIC_SynHdecor5
hdecor[5] signal

enumerator kXBIC_SynHdecor4
hdecor[4] signal

enumerator kXBIC_SynHdecor3
hdecor[3] signal

enumerator kXBIC_SynHdecor2
hdecor[2] signal

enumerator kXBIC_SynHdecor1
hdecor[1] signal

enumerator kXBIC_SynHdecor0
hdecor[0] signal

typedef enum *_xbic_error_syndromes* xbic_error_syndromes_t
_xbic_error_syndromes XBIC error syndromes.

2.69 XRDC: Extended Resource Domain Controller

void XRDC_GetHardwareConfig(XRDC_Type *base, *xrdc_hardware_config_t* *config)
Gets the XRDC hardware configuration.

This function gets the XRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

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

static inline void XRDC_LockGlobalControl(XRDC_Type *base)
Locks the XRDC global control register XRDC_CR.

This function locks the XRDC_CR register. After it is locked, the register is read-only until the next reset.

Parameters

- base – XRDC peripheral base address.

static inline void XRDC_SetGlobalValid(XRDC_Type *base, bool valid)
Sets the XRDC global valid.

This function sets the XRDC global valid or invalid. When the XRDC is global invalid, all accesses from all bus masters to all slaves are allowed.

Parameters

- base – XRDC peripheral base address.
- valid – True to valid XRDC.

static inline uint8_t XRDC_GetCurrentMasterDomainId(XRDC_Type *base)

Gets the domain ID of the current bus master.

This function returns the domain ID of the current bus master.

Parameters

- base – XRDC peripheral base address.

Returns

Domain ID of current bus master.

status_t XRDC_GetAndClearFirstDomainError(XRDC_Type *base, *xrdc_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 – XRDC 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_XRDC_NoError.

status_t XRDC_GetAndClearFirstSpecificDomainError(XRDC_Type *base, *xrdc_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 – XRDC 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_XRDC_NoError.

void XRDC_GetPidDefaultConfig(*xrdc_pid_config_t* *config)

Gets the default PID configuration structure.

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

```
config->pid       = 0U;
config->tsmEnable = 0U;
config->sp4smEnable = 0U;
config->lockMode  = kXRDC_PidLockSecurePrivilegeWritable;
```

Parameters

- config – Pointer to the configuration structure.

void XRDC_SetPidConfig(XRDC_Type *base, xrdc_master_t master, const *xrdc_pid_config_t* *config)

Configures the PID for a specific bus master.

This function configures the PID for a specific bus master. Do not use this function for non-processor bus masters.

Parameters

- base – XRDC peripheral base address.
- master – Which bus master to configure.
- config – Pointer to the configuration structure.

static inline void XRDC_SetPidLockMode(XRDC_Type *base, xrdc_master_t master, *xrdc_pid_lock_t* lockMode)

Sets the PID configuration register lock mode.

This function sets the PID configuration register lock XRDC_PIDn[LK2].

Parameters

- base – XRDC peripheral base address.
- master – Which master's PID to lock.
- lockMode – Lock mode to set.

void XRDC_GetDefaultNonProcessorDomainAssignment(*xrdc_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       = kXRDC_ForceUser;
assignment->privilegeAttr       = kXRDC_ForceSecure;
assignment->bypassDomainId      = 0U;
assignment->blogicPartId        = 0U;
assignment->benableLogicPartId  = 0U;
assignment->lock                = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void XRDC_GetDefaultProcessorDomainAssignment(*xrdc_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      = kXRDC_DidMda;
assignment->dpidEnable          = kXRDC_PidDisable;
assignment->pidMask             = 0U;
assignment->pid                 = 0U;
assignment->logicPartId         = 0U;
assignment->enableLogicPartId   = 0U;
assignment->lock                = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void XRDC_SetNonProcessorDomainAssignment(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex, const *xrdc_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.

```
xrdc_non_processor_domain_assignment_t nonProcessorAssignment;

XRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Which assignment register to set.
- domainAssignment – Pointer to the assignment structure.

void XRDC_SetProcessorDomainAssignment(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex, const *xrdc_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. In this example, there are 3 assignment registers for core 0.

```
xrdc_processor_domain_assignment_t processorAssignment;

XRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 1;
processorAssignment.xxx      = xxx;
XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterCpu0, 0U, &processorAssignment);

processorAssignment.domainId = 2;
processorAssignment.xxx      = xxx;
XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterCpu0, 1U, &processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
XRDC_SetMasterDomainAssignment(XRDC, kXrdcMasterCpu0, 2U, &processorAssignment);
```

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Which assignment register to set.
- domainAssignment – Pointer to the assignment structure.

static inline void XRDC_LockMasterDomainAssignment(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex)

Locks the bus master domain assignment register.

This function locks the master domain assignment. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to lock. After it is locked, the register can't be changed until next reset.

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Which assignment register to lock.

static inline void XRDC_SetMasterDomainAssignmentValid(XRDC_Type *base, xrdc_master_t master, uint8_t assignIndex, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to configure.

Parameters

- base – XRDC peripheral base address.
- master – Which master to configure.
- assignIndex – Index for the domain assignment register.
- valid – True to set valid, false to set invalid.

void XRDC_GetMemAccessDefaultConfig(*xrdc_mem_access_config_t* *config)

Gets the default memory region access policy.

This function gets the default memory region access policy. It sets the policy as follows:

```
config->enableSema          = false;
config->semaNum             = 0U;
config->subRegionDisableMask = 0U;
config->size                = kXrdcMemSizeNone;
config->lockMode            = kXRDC_AccessConfigLockWritable;
config->baseAddress         = 0U;
config->policy[0]           = kXRDC_AccessPolicyNone;
config->policy[1]           = kXRDC_AccessPolicyNone;
...
config->policy[15]          = kXRDC_AccessPolicyNone;
```

Parameters

- config – Pointer to the configuration structure.

void XRDC_SetMemAccessConfig(XRDC_Type *base, const *xrdc_mem_access_config_t* *config)

Sets the memory region access policy.

This function sets the memory region access configuration as valid. There are two methods to use it:

Example 1: Set one configuration run time. Set memory region 0x20000000 ~ 0x20000400 accessible by domain 0, use MRC0_1.

```
xrdc_mem_access_config_t config =
{
    .mem          = kXRDC_MemMrc0_1,
    .baseAddress = 0x20000000U,
    .size         = kXRDC_MemSize1K,
    .policy[0]    = kXRDC_AccessPolicyAll
};
XRDC_SetMemAccessConfig(XRDC, &config);
```

Example 2: Set multiple configurations during startup. Set memory region 0x20000000 ~ 0x20000400 accessible by domain 0, use MRC0_1. Set memory region 0x1FFF0000 ~ 0x1FFF0800 accessible by domain 0, use MRC0_2.

```
xrdc_mem_access_config_t configs[] =
{
    {
        .mem          = kXRDC_MemMrc0_1,
        .baseAddress = 0x20000000U,
        .size         = kXRDC_MemSize1K,
        .policy[0]    = kXRDC_AccessPolicyAll
    },
    {
        .mem          = kXRDC_MemMrc0_2,
        .baseAddress = 0x1FFF0000U,
        .size         = kXRDC_MemSize2K,
        .policy[0]    = kXRDC_AccessPolicyAll
    }
};

for (i=0U; i<((sizeof(configs)/sizeof(configs[0]))); i++)
{
    XRDC_SetMemAccessConfig(XRDC, &configs[i]);
}
```

Parameters

- base – XRDC peripheral base address.
- config – Pointer to the access policy configuration structure.

static inline void XRDC_SetMemAccessLockMode(XRDC_Type *base, xrdc_mem_t mem, *xrdc_access_config_lock_t* lockMode)

Sets the memory region descriptor register lock mode.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region descriptor to lock.
- lockMode – The lock mode to set.

static inline void XRDC_SetMemAccessValid(XRDC_Type *base, xrdc_mem_t mem, bool valid)

Sets the memory region descriptor as valid or invalid.

This function sets the memory region access configuration dynamically. For example:

```
xrdc_mem_access_config_t config =
{
    .mem          = kXRDC_MemMrc0_1,
    .baseAddress = 0x20000000U,
```

(continues on next page)

(continued from previous page)

```
    .size       = kXRDC_MemSize1K,
    .policy[0]  = kXRDC_AccessPolicyAll
};
XRDC_SetMemAccessConfig(XRDC, &config);

XRDC_SetMemAccessValid(kXRDC_MemMrc0_1, false);

XRDC_SetMemAccessValid(kXRDC_MemMrc0_1, true);
```

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region descriptor to set.
- valid – True to set valid, false to set invalid.

void XRDC_SetMemExclAccessLockMode(XRDC_Type *base, xrdc_mem_t mem, *xrdc_excl_access_lock_config_t* lockMode)

Sets the memory region exclusive access lock mode configuration.

Note: Any write to MRGD_W[0-3]_n clears the MRGD_W4_n[VLD] indicator so a coherent register state can be supported. It is indispensable to re-assert the valid bit when dynamically changing the EAL in the MRGD, which is done in this API.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region's exclusive access lock mode to configure.
- lockMode – The exclusive access lock mode to set.

void XRDC_ForceMemExclAccessLockRelease(XRDC_Type *base, xrdc_mem_t mem)

Forces the release of the memory region exclusive access lock.

A lock can be forced to the available state (EAL=10) by a domain that does not own the lock through the forced lock release procedure: The procedure to force a exclusive access lock release is as follows:

a. Write 0x02000046 to W1 register (PAC/MSC) or W3 register (MRC)
b. Write 0x02000052 to W1 register (PAC/MSC) or W3 register (MRC)

Note: The two writes must be consecutive, any intervening write to the register resets the sequence.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region's exclusive access lock to force release.

static inline uint8_t XRDC_GetMemExclAccessLockDomainOwner(XRDC_Type *base, xrdc_mem_t mem)

Gets the exclusive access lock domain owner of the memory region.

This function returns the domain ID of the exclusive access lock owner of the memory region.

Parameters

- base – XRDC peripheral base address.
- mem – Which memory region's exclusive access lock domain owner to get.

Returns

Domain ID of the memory region exclusive access lock owner.

void XRDC_GetPeriphAccessDefaultConfig(*xrdc_periph_access_config_t* *config)

Gets the default peripheral access configuration.

The default configuration is set as follows:

```
config->enableSema      = false;
config->semaNum         = 0U;
config->lockMode        = kXRDC_AccessConfigLockWritable;
config->policy[0]       = kXRDC_AccessPolicyNone;
config->policy[1]       = kXRDC_AccessPolicyNone;
...
config->policy[15]      = kXRDC_AccessPolicyNone;
```

Parameters

- config – Pointer to the configuration structure.

void XRDC_SetPeriphAccessConfig(XRDC_Type *base, const *xrdc_periph_access_config_t* *config)

Sets the peripheral access configuration.

This function sets the peripheral access configuration as valid. Two methods to use it: Method 1: Set for one peripheral, which is used for runtime settings. Example: set LPTMR0 accessible by domain 0

```
xrdc_periph_access_config_t config;

config.periph    = kXRDC_PeriphLptmr0;
config.policy[0] = kXRDC_AccessPolicyAll;
XRDC_SetPeriphAccessConfig(XRDC, &config);
```

Method 2: Set for multiple peripherals, which is used for initialization settings.

```
xrdc_periph_access_config_t configs[] =
{
    {
        .periph    = kXRDC_PeriphLptmr0,
        .policy[0] = kXRDC_AccessPolicyAll,
        .policy[1] = kXRDC_AccessPolicyAll
    },
    {
        .periph    = kXRDC_PeriphLpuart0,
        .policy[0] = kXRDC_AccessPolicyAll,
        .policy[1] = kXRDC_AccessPolicyAll
    }
};

for (i=0U; i<(sizeof(configs)/sizeof(configs[0])), i++)
{
    XRDC_SetPeriphAccessConfig(XRDC, &config[i]);
}
```

Parameters

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

static inline void XRDC_SetPeriphAccessLockMode(XRDC_Type *base, xrdc_periph_t periph, *xrdc_access_config_lock_t* lockMode)

Sets the peripheral access configuration register lock mode.

Parameters

- base – XRDC peripheral base address.

- periph – Which peripheral access configuration register to lock.
- lockMode – The lock mode to set.

static inline void XRDC_SetPeriphAccessValid(XRDC_Type *base, xrdc_periph_t periph, bool valid)

Sets the peripheral access as valid or invalid.

This function sets the peripheral access configuration dynamically. For example:

```
xrdc_periph_access_config_t config =
{
    .periph    = kXRDC_PeriphLptmr0;
    .policy[0] = kXRDC_AccessPolicyAll;
};
XRDC_SetPeriphAccessConfig(XRDC, &config);

XRDC_SetPeriphAccessValid(kXrdcPeriLptmr0, false);

XRDC_SetPeriphAccessValid(kXrdcPeriLptmr0, true);
```

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral access configuration to set.
- valid – True to set valid, false to set invalid.

static inline void XRDC_SetPeriphExclAccessLockMode(XRDC_Type *base, xrdc_periph_t periph, *xrdc_excl_access_lock_config_t* lockMode)

Sets the peripheral exclusive access lock mode configuration.

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral's exclusive access lock mode to configure.
- lockMode – The exclusive access lock mode to set.

void XRDC_ForcePeriphExclAccessLockRelease(XRDC_Type *base, xrdc_periph_t periph)

Forces the release of the peripheral exclusive access lock.

A lock can be forced to the available state (EAL=10) by a domain that does not own the lock through the forced lock release procedure: The procedure to force a exclusive access lock release is as follows:

a. Write 0x02000046 to W1 register (PAC/MSC) or W3 register (MRC)
b. Write 0x02000052 to W1 register (PAC/MSC) or W3 register (MRC)

Note: The two writes must be consecutive, any intervening write to the register resets the sequence.

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral's exclusive access lock to force release.

static inline uint8_t XRDC_GetPeriphExclAccessLockDomainOwner(XRDC_Type *base, xrdc_periph_t periph)

Gets the exclusive access lock domain owner of the peripheral.

This function returns the domain ID of the exclusive access lock owner of the peripheral.

Parameters

- base – XRDC peripheral base address.
- periph – Which peripheral's exclusive access lock domain owner to get.

Returns
Domain ID of the peripheral exclusive access lock owner.

XRDC status _xrdc_status.

Values:

enumerator kStatus_XRDC_NoError
No error captured.

enum _xrdc_pid_enable
XRDC PID enable mode, the register bit XRDC_MDA_Wx[PE], used for domain hit evaluation.

Values:

enumerator kXRDC_PidDisable
PID is not used in domain hit evalution.

enumerator kXRDC_PidDisable1
PID is not used in domain hit evalution.

enumerator kXRDC_PidExp0
((XRDC_MDA_W[PID] & ~XRDC_MDA_W[PIDM]) == (XRDC_PID[PID] & ~XRDC_MDA_W[PIDM])).

enumerator kXRDC_PidExp1
~((XRDC_MDA_W[PID] & ~XRDC_MDA_W[PIDM]) == (XRDC_PID[PID] & ~XRDC_MDA_W[PIDM])).

enum _xrdc_did_sel
XRDC domain ID select method, the register bit XRDC_MDA_Wx[DIDS], used for domain hit evaluation.

Values:

enumerator kXRDC_DidMda
Use MDAn[3:0] as DID.

enumerator kXRDC_DidInput
Use the input DID (DID_in) as DID.

enumerator kXRDC_DidMdaAndInput
Use MDAn[3:2] concatenated with DID_in[1:0] as DID.

enumerator kXRDC_DidReserved
Reserved.

enum _xrdc_secure_attr
XRDC secure attribute, the register bit XRDC_MDA_Wx[SA], used for non-processor bus master domain assignment.

Values:

enumerator kXRDC_ForceSecure
Force the bus attribute for this master to secure.

enumerator kXRDC_ForceNonSecure
Force the bus attribute for this master to non-secure.

enumerator kXRDC_MasterSecure
Use the bus master's secure/nonsecure attribute directly.

enumerator kXRDC_MasterSecure1
Use the bus master's secure/nonsecure attribute directly.

enum _xrdc_privilege_attr
XRDC privileged attribute, the register bit XRDC_MDA_Wx[PA], used for non-processor bus master domain assignment.

Values:

enumerator kXRDC_ForceUser
Force the bus attribute for this master to user.

enumerator kXRDC_ForcePrivilege
Force the bus attribute for this master to privileged.

enumerator kXRDC_MasterPrivilege
Use the bus master's attribute directly.

enumerator kXRDC_MasterPrivilege1
Use the bus master's attribute directly.

enum _xrdc_pid_lock
XRDC PID LK2 definition XRDC_PIDn[LK2].

Values:

enumerator kXRDC_PidLockSecurePrivilegeWritable
Writable by any secure privileged write.

enumerator kXRDC_PidLockSecurePrivilegeWritable1
Writable by any secure privileged write.

enumerator kXRDC_PidLockMasterXOnly
PIDx is only writable by master x.

enumerator kXRDC_PidLockLocked
Read-only until the next reset.

enum _xrdc_access_policy
XRDC domain access control policy.

Values:

enumerator kXRDC_AccessPolicyNone

enumerator kXRDC_AccessPolicySpuR

enumerator kXRDC_AccessPolicySpRw

enumerator kXRDC_AccessPolicySpuRw

enumerator kXRDC_AccessPolicySpuRwNpR

enumerator kXRDC_AccessPolicySpuRwNpuR

enumerator kXRDC_AccessPolicySpuRwNpRw

enumerator kXRDC_AccessPolicyAll

enum _xrdc_access_config_lock
Access configuration lock mode, the register field PDAC and MRGD LK2.

Values:

enumerator kXRDC_AccessConfigLockWritable
Entire PDACn/MRGDn can be written.

enumerator kXRDC_AccessConfigLockWritable1
Entire PDACn/MRGDn can be written.

enumerator kXRDC_AccessConfigLockDomainXOnly
Domain x only write the DxACP field.

enumerator kXRDC_AccessConfigLockLocked
PDACn is read-only until the next reset.

enum _xrdc_excl_access_lock_config
Exclusive access lock mode configuration, the register field PDAC and MRGD EAL.

Values:

enumerator kXRDC_ExclAccessLockDisabled
Lock disabled.

enumerator kXRDC_ExclAccessLockDisabledUntilNextRst
Lock disabled until next reset.

enumerator kXRDC_ExclAccessLockEnabledStateAvail
Lock enabled, lock state = available.

enumerator kXRDC_ExclAccessLockEnabledStateNotAvail
Lock enabled, lock state = not available.

enum _xrdc_mem_code_region
XRDC memory code region indicator.

Values:

enumerator kXRDC_MemCodeRegion0
Code region indicator 0=data.

enumerator kXRDC_MemCodeRegion1
Code region indicator 1=code.

enum _xrdc_access_flags_select
XRDC domain access flags/policy select.

Policy: {R,W,X} Read, write, execute flags. flag = 0 : inhibits access, flag = 1 : allows access. policy => SecurePriv_NonSecurePriv_SecureUser_NonSecureUsr xxx_xxx_xxx_xxx => PS{R,W,X}_PN{R,W,X}_US{R,W,X}_UN{R,W,X}

```
PS > PN > US > UN        PS > PN > US > UN
```

DxSEL CodeRegion = 0 CodeRegion = 1 000 000_000_000_000 = 0x000 000_000_000_000 = 0x000 001 ACCSET1 010 ACCSET2 011 110_000_000_000 = 0xC00 001_001_001_001 = 0x249 100 110_110_000_000 = 0xD80 111_000_000_000 = 0xE00 101 110_110_100_100 = 0xDA4 110_111_000_000 = 0xDC0 110 110_110_110_000 = 0xDB0 110_110_111_000 = 0xDB8 111 110_110_110_110 = 0xDB6 110_110_111_111 = 0xDBF

Values:

enumerator kXRDC_AccessFlagsNone

enumerator kXRDC_AccessFlagsAlt1

enumerator kXRDC_AccessFlagsAlt2

enumerator kXRDC_AccessFlagsAlt3

enumerator kXRDC_AccessFlagsAlt4

enumerator kXRDC_AccessFlagsAlt5

enumerator kXRDC_AccessFlagsAlt6

enumerator kXRDC_AccessFlagsAlt7

enum _xrdc_controller

XRDC controller definition for domain error check.

Values:

enumerator kXRDC_MemController0

Memory region controller 0.

enumerator kXRDC_MemController1

Memory region controller 1.

enumerator kXRDC_MemController2

Memory region controller 2.

enumerator kXRDC_MemController3

Memory region controller 3.

enumerator kXRDC_MemController4

Memory region controller 4.

enumerator kXRDC_MemController5

Memory region controller 5.

enumerator kXRDC_MemController6

Memory region controller 6.

enumerator kXRDC_MemController7

Memory region controller 7.

enumerator kXRDC_MemController8

Memory region controller 8.

enumerator kXRDC_MemController9

Memory region controller 9.

enumerator kXRDC_MemController10

Memory region controller 10.

enumerator kXRDC_MemController11

Memory region controller 11.

enumerator kXRDC_MemController12

Memory region controller 12.

enumerator kXRDC_MemController13

Memory region controller 13.

enumerator kXRDC_MemController14

Memory region controller 14.

enumerator kXRDC_MemController15

Memory region controller 15.

enumerator kXRDC_PeriphController0

Peripheral access controller 0.

enumerator kXRDC_PeriphController1
Peripheral access controller 1.

enumerator kXRDC_PeriphController2
Peripheral access controller 2.

enumerator kXRDC_PeriphController3
Peripheral access controller 3.

enum _xrdc_error_state
XRDC domain error state definition XRDC_DERR_W1_n[EST].

Values:

enumerator kXRDC_ErrorStateNone
No access violation detected.

enumerator kXRDC_ErrorStateNone1
No access violation detected.

enumerator kXRDC_ErrorStateSingle
Single access violation detected.

enumerator kXRDC_ErrorStateMulti
Multiple access violation detected.

enum _xrdc_error_attr
XRDC domain error attribute definition XRDC_DERR_W1_n[EATR].

Values:

enumerator kXRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kXRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kXRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kXRDC_ErrorSecurePrivilegeData
Secure privileged mode, data access.

enumerator kXRDC_ErrorNonSecureUserInst
NonSecure user mode, instruction fetch access.

enumerator kXRDC_ErrorNonSecureUserData
NonSecure user mode, data access.

enumerator kXRDC_ErrorNonSecurePrivilegeInst
NonSecure privileged mode, instruction fetch access.

enumerator kXRDC_ErrorNonSecurePrivilegeData
NonSecure privileged mode, data access.

enum _xrdc_error_type
XRDC domain error access type definition XRDC_DERR_W1_n[ERW].

Values:

enumerator kXRDC_ErrorTypeRead
Error occurs on read reference.

enumerator kXRDC_ErrorTypeWrite
Error occurs on write reference.

typedef struct *_xrdc_hardware_config* xrdc_hardware_config_t
XRDC hardware configuration.

typedef enum *_xrdc_pid_enable* xrdc_pid_enable_t
XRDC PID enable mode, the register bit XRDC_MDA_Wx[PE], used for domain hit evaluation.

typedef enum *_xrdc_did_sel* xrdc_did_sel_t
XRDC domain ID select method, the register bit XRDC_MDA_Wx[DIDS], used for domain hit evaluation.

typedef enum *_xrdc_secure_attr* xrdc_secure_attr_t
XRDC secure attribute, the register bit XRDC_MDA_Wx[SA], used for non-processor bus master domain assignment.

typedef enum *_xrdc_privilege_attr* xrdc_privilege_attr_t
XRDC privileged attribute, the register bit XRDC_MDA_Wx[PA], used for non-processor bus master domain assignment.

typedef struct *_xrdc_processor_domain_assignment* xrdc_processor_domain_assignment_t
Domain assignment for the processor bus master.

typedef struct *_xrdc_non_processor_domain_assignment*
xrdc_non_processor_domain_assignment_t
Domain assignment for the non-processor bus master.

typedef enum *_xrdc_pid_lock* xrdc_pid_lock_t
XRDC PID LK2 definition XRDC_PIDn[LK2].

typedef struct *_xrdc_pid_config* xrdc_pid_config_t
XRDC process identifier (PID) configuration.

typedef enum *_xrdc_access_policy* xrdc_access_policy_t
XRDC domain access control policy.

typedef enum *_xrdc_access_config_lock* xrdc_access_config_lock_t
Access configuration lock mode, the register field PDAC and MRGD LK2.

typedef enum *_xrdc_excl_access_lock_config* xrdc_excl_access_lock_config_t
Exclusive access lock mode configuration, the register field PDAC and MRGD EAL.

typedef struct *_xrdc_periph_access_config* xrdc_periph_access_config_t
XRDC peripheral domain access control configuration.

typedef enum *_xrdc_mem_code_region* xrdc_mem_code_region_t
XRDC memory code region indicator.

typedef enum *_xrdc_access_flags_select* xrdc_access_flags_select_t
XRDC domain access flags/policy select.

Policy: {R,W,X} Read, write, execute flags. flag = 0 : inhibits access, flag = 1 : allows access. policy => SecurePriv_NonSecurePriv_SecureUser_NonSecureUsr xxx_xxx_xxx_xxx => PS{R,W,X}_PN{R,W,X}_US{R,W,X}_UN{R,W,X}

```
PS > PN > US > UN        PS > PN > US > UN
```

DxSEL CodeRegion = 0 CodeRegion = 1 000 000_000_000_000 = 0x000 000_000_000_000 = 0x000 001 ACCSET1 010 ACCSET2 011 110_000_000_000 = 0xC00 001_001_001_001 = 0x249 100 110_110_000_000 = 0xD80 111_000_000_000 = 0xE00 101 110_110_100_100 = 0xDA4 110_111_000_000 = 0xDC0 110 110_110_110_000 = 0xDB0 110_110_111_000 = 0xDB8 111 110_110_110_110 = 0xDB6 110_110_111_111 = 0xDBF

typedef struct *_xrdc_mem_access_config* xrdc_mem_access_config_t
XRDC memory region domain access control configuration.

typedef enum *_xrdc_controller* xrdc_controller_t
XRDC controller definition for domain error check.

typedef enum *_xrdc_error_state* xrdc_error_state_t
XRDC domain error state definition XRDC_DERR_W1_n[EST].

typedef enum *_xrdc_error_attr* xrdc_error_attr_t
XRDC domain error attribute definition XRDC_DERR_W1_n[EATR].

typedef enum *_xrdc_error_type* xrdc_error_type_t
XRDC domain error access type definition XRDC_DERR_W1_n[ERW].

typedef struct *_xrdc_error* xrdc_error_t
XRDC domain error definition.

void XRDC_Init(XRDC_Type *base)
Initializes the XRDC module.

This function enables the XRDC clock.

Parameters

- base – XRDC peripheral base address.

void XRDC_Deinit(XRDC_Type *base)
De-initializes the XRDC module.

This function disables the XRDC clock.

Parameters

- base – XRDC peripheral base address.

FSL_XRDC_DRIVER_VERSION

struct _xrdc_hardware_config
#include <fsl_xrdc.h> XRDC hardware configuration.

Public Members

uint8_t masterNumber
Number of bus masters.

uint8_t domainNumber
Number of domains.

uint8_t pacNumber
Number of PACs.

uint8_t mrcNumber
Number of MRCs.

struct _xrdc_processor_domain_assignment
#include <fsl_xrdc.h> Domain assignment for the processor bus master.

Public Members

uint32_t domainId
Domain ID.

uint32_t domainIdSelect
: Domain ID select method, see xrdc_did_sel_t.

uint32_t pidEnable
: PId enable method, see xrdc_pid_enable_t.

uint32_t pidMask
: PId mask.

uint32_t ___pad0___
: Reserved.

uint32_t pid
: PId value.

uint32_t ___pad1___
: Reserved.

uint32_t ___pad2___
: Reserved.

uint32_t ___pad3___
: Reserved.

uint32_t ___pad4___
: Reserved.

uint32_t lock
: Lock the register.

uint32_t ___pad5___
: Reserved.

struct __xrdc__non__processor__domain__assignment
: *#include <fsl_xrdc.h>* Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId
: Domain ID.

uint32_t privilegeAttr
: Privileged attribute, see xrdc_privilege_attr_t.

uint32_t secureAttr
: Secure attribute, see xrdc_secure_attr_t.

uint32_t bypassDomainId
: Bypass domain ID.

uint32_t ___pad0___
: Reserved.

uint32_t ___pad1___
: Reserved.

uint32_t ___pad2___
: Reserved.

uint32_t ___pad3___
: Reserved.

uint32_t lock
Lock the register.

uint32_t ___pad4___
Reserved.

struct __xrdc__pid__config
#include <fsl_xrdc.h> XRDC process identifier (PID) configuration.

Public Members

uint32_t pid
PID value, PIDn[PID].

uint32_t ___pad0___
Reserved.

uint32_t tsmEnable
Enable three-state model.

uint32_t lockMode
PIDn configuration lock mode, see xrdc_pid_lock_t.

uint32_t ___pad1___
Reserved.

struct __xrdc__periph__access__config
#include <fsl_xrdc.h> XRDC peripheral domain access control configuration.

Public Members

xrdc_periph_t periph
Peripheral name.

xrdc_access_config_lock_t lockMode
PDACn lock configuration.

xrdc_excl_access_lock_config_t exclAccessLockMode
Exclusive access lock configuration.

xrdc_access_policy_t policy[1]
Access policy for each domain.

struct __xrdc__mem__access__config
#include <fsl_xrdc.h> XRDC memory region domain access control configuration.

Public Members

xrdc_mem_t mem
Memory region descriptor name.

xrdc_access_config_lock_t lockMode
MRGDn lock configuration.

xrdc_access_flags_select_t policy[1]
Access policy/flags select for each domain.

xrdc_mem_code_region_t codeRegion
Code region select. xrdc_mem_code_region_t.

uint32_t baseAddress
: Memory region base/start address.

uint32_t endAddress
: Memory region end address. The 5 LSB of end address is ignored and forced to 0x1F by hardware.

xrdc_excl_access_lock_config_t exclAccessLockMode
: Exclusive access lock configuration.

struct __xrdc_error

#include <fsl_xrdc.h> XRDC domain error definition.

Public Members

xrdc_controller_t controller
: Which controller captured access violation.

uint32_t address
: Access address that generated access violation.

xrdc_error_state_t errorState
: Error state.

xrdc_error_attr_t errorAttr
: Error attribute.

xrdc_error_type_t errorType
: Error type.

uint8_t errorPort
: Error port.

uint8_t domainId
: Domain ID.

Chapter 3

Middleware

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