MCUXpresso SDK Documentation

Release 25.12.00-pvw2



NXP
Nov 14, 2025

Table of contents

1 MIMXRT700-EVK **3**
- 1.1 Overview . . . 3
- 1.2 Getting Started with MCUXpresso SDK Package . . . 3
  - 1.2.1 Getting Started with Package . . . 3
- 1.3 Getting Started with MCUXpresso SDK GitHub . . . 35
  - 1.3.1 Getting Started with MCUXpresso SDK Repository . . . 35
- 1.4 Getting Started with MCUXpresso SDK Xplorer . . . 48
  - 1.4.1 Getting Started with MCUXpresso SDK Xplorer . . . 48
- 1.5 Release Notes . . . 65
  - 1.5.1 MCUXpresso SDK Release Notes . . . 65
- 1.6 ChangeLog . . . 70
  - 1.6.1 MCUXpresso SDK Changelog . . . 70
- 1.7 Driver API Reference Manual . . . 168
- 1.8 Middleware Documentation . . . 168
  - 1.8.1 VG-Lite GPU Library . . . 169
  - 1.8.2 Multicore . . . 169
  - 1.8.3 MCU Boot . . . 169
  - 1.8.4 eIQ . . . 169
  - 1.8.5 FreeMASTER . . . 169
  - 1.8.6 AWS IoT . . . 169
  - 1.8.7 NXP Wi-Fi . . . 169
  - 1.8.8 FreeRTOS . . . 169
  - 1.8.9 Wireless EdgeFast Bluetooth PAL . . . 169
  - 1.8.10 lwIP . . . 169
  - 1.8.11 File systemFatfs . . . 169
  - 1.8.12 DSP Audio Streamer . . . 170

2 MIMXRT798S **171**
- 2.1 ACMP: Analog Comparator Driver . . . 171
- 2.2 CACHE: CACHE Memory Controller . . . 177
- 2.3 CDOG . . . 181
- 2.4 Clock Driver . . . 185
- 2.5 CRC: Cyclic Redundancy Check Driver . . . 232
- 2.6 CTIMER: Standard counter/timers . . . 234
- 2.7 Dsp . . . 244
- 2.8 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 245
- 2.9 eDMA core Driver . . . 276
- 2.10 eDMA soc Driver . . . 283
- 2.11 Ezhv . . . 284
- 2.12 FGPIO Driver . . . 287
- 2.13 FlexIO: FlexIO Driver . . . 287
- 2.14 FlexIO Driver . . . 287
- 2.15 FlexIO eDMA I2S Driver . . . 305
- 2.16 FlexIO eDMA SPI Driver . . . 308
- 2.17 FlexIO eDMA UART Driver . . . 311
- 2.18 FlexIO I2C Master Driver . . . 314

2.19 FlexIO I2S Driver . . . 323
2.20 FlexIO SPI Driver . . . 333
2.21 FlexIO UART Driver . . . 346
2.22 FREQME: Frequency Measurement . . . 357
2.23 GDET . . . 357
2.24 Glitch Detect . . . 358
2.25 GLIKEY . . . 358
2.26 GLIKEY . . . 358
2.27 GPIO: General-Purpose Input/Output Driver . . . 363
2.28 GPIO Driver . . . 366
2.29 I3C: I3C Driver . . . 371
2.30 I3C Common Driver . . . 374
2.31 I3C Master Driver . . . 376
2.32 I3C Master DMA Driver . . . 402
2.33 I3C Slave Driver . . . 405
2.34 I3C Slave DMA Driver . . . 417
2.35 INPUTMUX: Input Multiplexing Driver . . . 420
2.36 Iopctl_driver . . . 429
2.37 IRTC: IRTC Driver . . . 431
2.38 Intrusion and Tamper Response Controller . . . 439
2.39 ITRC . . . 439
2.40 JPEGDEC: JPEG decoder Driver . . . 442
2.41 Common Driver . . . 450
2.42 LCDIF: LCD interface . . . 462
2.43 LPADC: 12-bit SAR Analog-to-Digital Converter Driver . . . 484
2.44 Lpc_freqme . . . 503
2.45 LPFLEXCOMM: LPFLEXCOMM Driver . . . 509
2.46 LPFLEXCOMM Driver . . . 509
2.47 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 510
2.48 LPI2C Master Driver . . . 510
2.49 LPI2C Master DMA Driver . . . 524
2.50 LPI2C Slave Driver . . . 527
2.51 LPSPI: Low Power Serial Peripheral Interface . . . 537
2.52 LPSPI Peripheral driver . . . 537
2.53 LPSPI eDMA Driver . . . 559
2.54 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 564
2.55 LPUART Driver . . . 564
2.56 LPUART eDMA Driver . . . 585
2.57 MIPI DSI Driver . . . 588
2.58 MIPI_DSI: MIPI DSI Host Controller . . . 606
2.59 MMU: Memory Map Unit . . . 606
2.60 MRT: Multi-Rate Timer . . . 618
2.61 MU: Messaging Unit . . . 622
2.62 This type defines status return values used by NBOOT functions that are not easily . . . 635
2.63 OSTIMER: OS Event Timer Driver . . . 636
2.64 PDM: Microphone Interface . . . 639
2.65 PDM Driver . . . 639
2.66 PDM EDMA Driver . . . 657
2.67 PINT: Pin Interrupt and Pattern Match Driver . . . 661
2.68 PNGDEC: PNG decoder Driver . . . 670
2.69 Power Driver . . . 674
2.70 PUF: Physical Unclonable Function . . . 698
2.71 Puf_v3_driver . . . 698
2.72 Reset Driver . . . 704
2.73 Runbootloader . . . 714
2.74 SAI: Serial Audio Interface . . . 715
2.75 SAI Driver . . . 715
2.76 SAI EDMA Driver . . . 741

2.77 Sbloader . . . 748
2.78 SCTimer: SCTimer/PWM (SCT) . . . 752
2.79 SDADC: SDADC Module . . . 768
2.80 SEMA42: Hardware Semaphores Driver . . . 783
2.81 Soc_mipi_dsi . . . 786
2.82 SYSPM: System Performance Monitor . . . 786
2.83 TRNG: True Random Number Generator . . . 789
2.84 USDHC: Ultra Secured Digital Host Controller Driver . . . 794
2.85 UTICK: MictoTick Timer Driver . . . 824
2.86 WWDT: Windowed Watchdog Timer Driver . . . 825
2.87 CACHE: CACHE Memory Controller . . . 829
2.88 XSPI: Serial Peripheral Interface Driver . . . 831
2.89 XSPI eDMA Driver . . . 885

3 Middleware **889**
3.1 Boot . . . 889
3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource . . . 889
3.1.2 MCUboot . . . 890
3.2 eIQ . . . 891
3.2.1 eIQ . . . 891
3.3 Motor Control . . . 919
3.3.1 FreeMASTER . . . 920
3.4 Wireless . . . 957
3.4.1 NXP Wireless Framework and Stacks . . . 957
3.4.2 EdgeFast Bluetooth . . . 957

4 RTOS **1033**
4.1 FreeRTOS . . . 1033
4.1.1 FreeRTOS kernel . . . 1033
4.1.2 FreeRTOS drivers . . . 1033
4.1.3 backoffalgorithm . . . 1033
4.1.4 corehttp . . . 1033
4.1.5 corejson . . . 1033
4.1.6 coremqtt . . . 1034
4.1.7 corepkcs11 . . . 1034
4.1.8 freertos-plus-tcp . . . 1034

This documentation contains information specific to the mimxrt700evk board.

Chapter 1

MIMXRT700-EVK

1.1 Overview

The i.MX RT700 EVK (MIMXRT700-EVK) features NXPs advanced implementation of the Arm Cortex-M33 core, combined with the highly optimized Cadence Tensilica HiFi4 and HiFi1 DSP processor cores. MIMXRT700-EVK supports development for the MIMXRT798, MIMXRT758 and MIMXRT735 products and its features make it ideal for portable HMI applications. The i.MX RT700 EVK can help jump start your next design with the included schematics and layout files.



MCU device and part on board is shown below:

- Device: MIMXRT798S
- PartNumber: MIMXRT798SGFOB

1.2 Getting Started with MCUXpresso SDK Package

1.2.1 Getting Started with Package

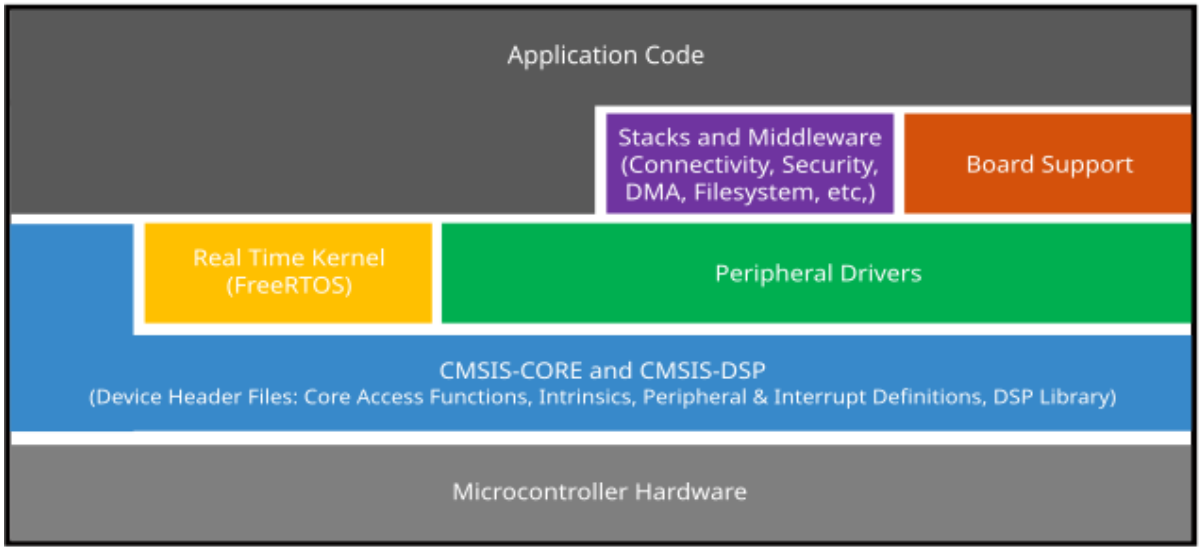
Overview

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

such as FreeRTOS and Azure RTOS,a USB host and device stack, and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for MIMXRT700-EVK* (document MCUXSDKRT798RN).

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):

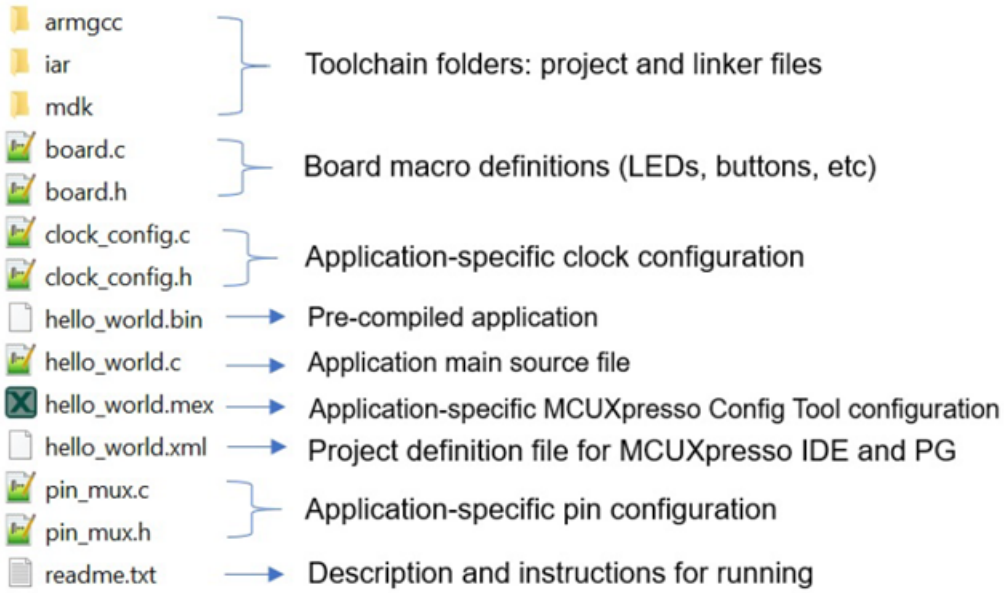
- cmsis_driver_examples: Simple applications intended to show how to use CMSIS drivers.
- demo_apps: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications that show how to use the MCUXpresso SDK 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 RTOS drivers.
- usb_examples: Applications that use the USB host/device/OTG stack.

Example application structure This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

Each <*board_name*> folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the hello_world example (part of

the *demo_apps* folder), the same general rules apply to any type of example in the <*board_name*> folder.

In the *hello_world* application folder, you see the following contents:



All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

Parent topic:*MCUXpresso SDK board support package folders*

Locating example application source files When opening an example application in any of the supported IDEs, 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 an RTOS, there are references to the appropriate source code. RTOSes are in the *rtos* folder. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

Parent topic:*MCUXpresso SDK board support package folders*

Run a demo using MCUXpresso IDE

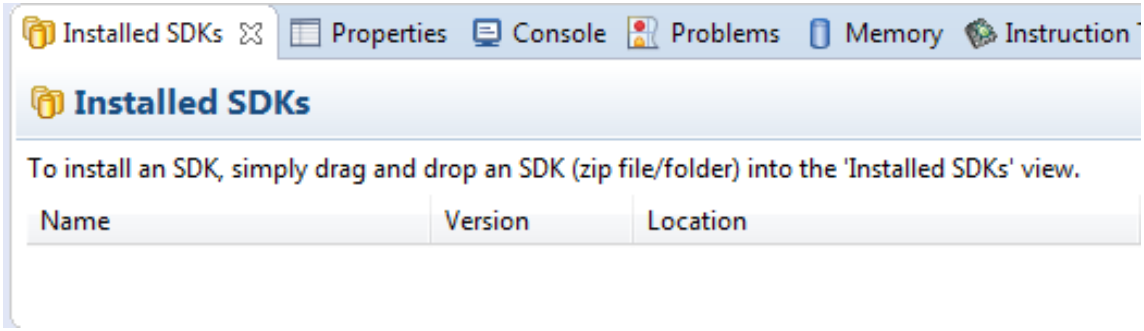
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello_world demo application targeted for the target hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

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 the 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 locate outside the MCUXpresso SDK tree.

Parent topic:*Run a demo using MCUXpresso IDE*

Build an example application To build an example application, perform the following steps.

1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK.

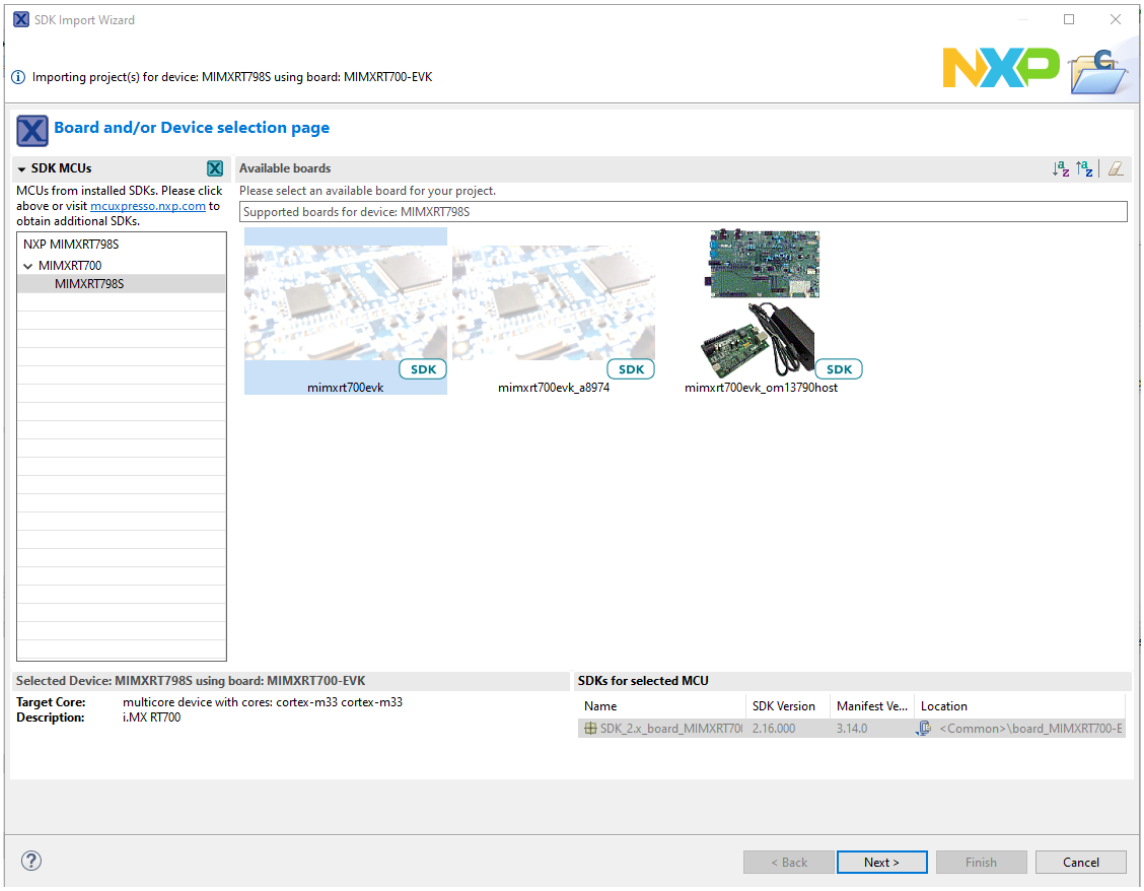


   In the window that appears, click **OK**and wait until the import has finished.

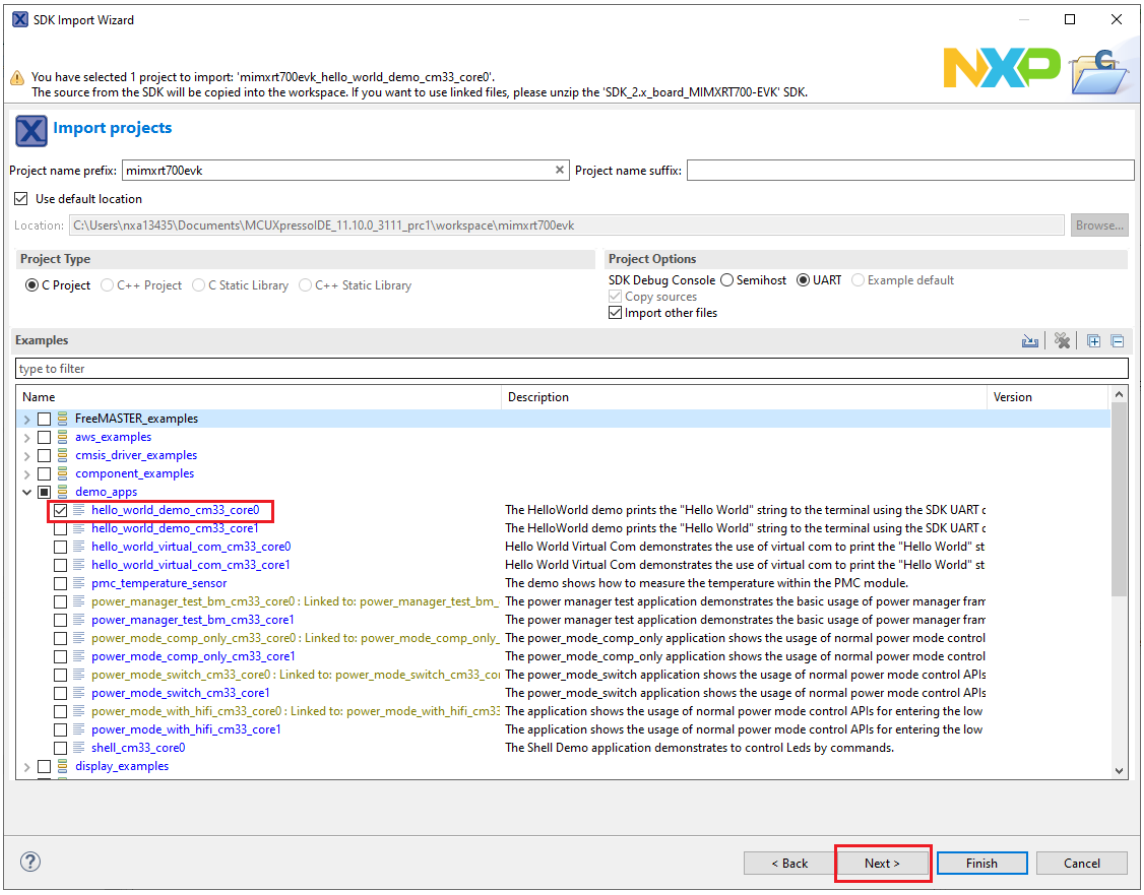
2. On the **Quickstart Panel**, click **Import SDK example(s)**.



3. In the window that appears, select **mimxrt700evk** or **kits** and click **Next**.



4. Expand the *demo_apps* folder and select **hello_world_demo_cm33_core0**. Then, click **Next**.

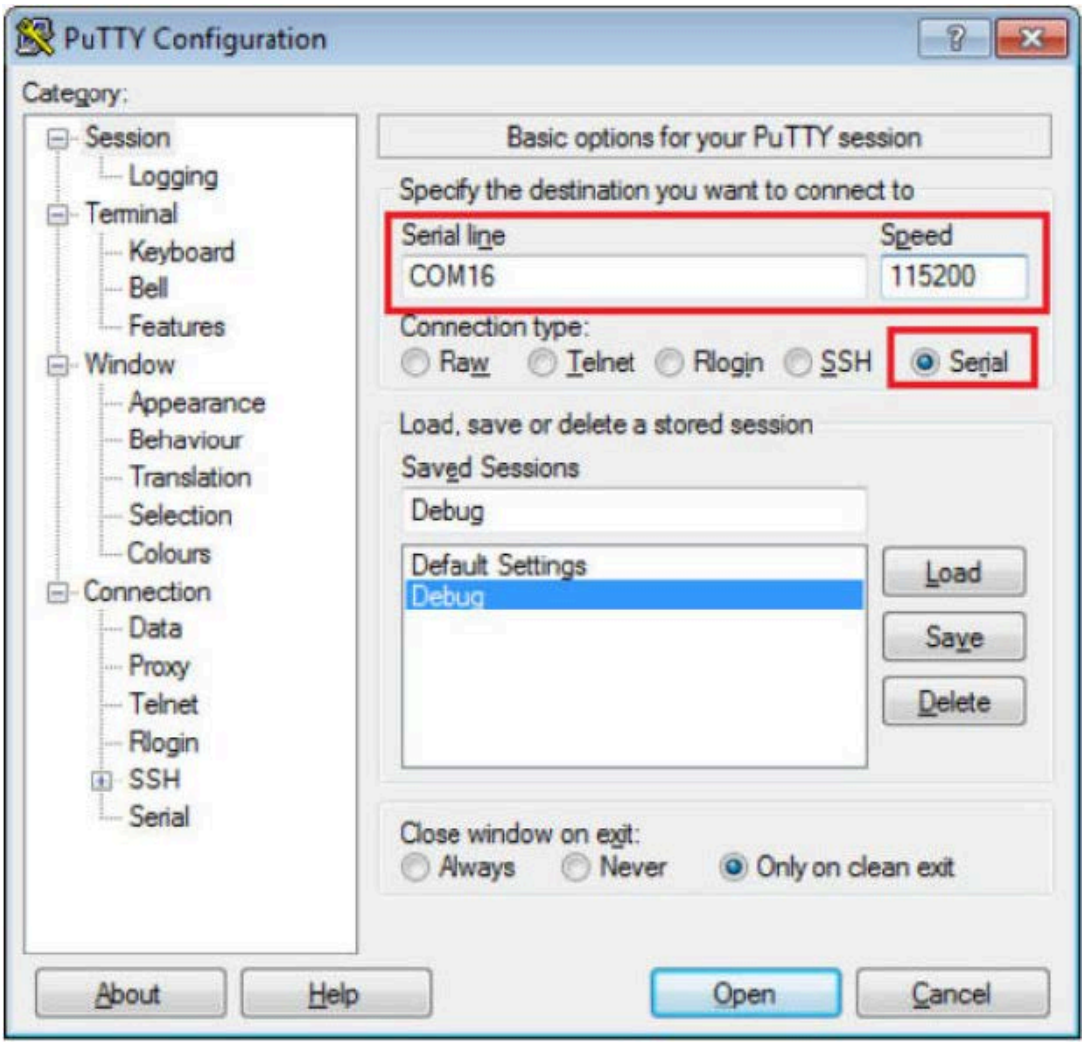


5. Click **Finish**.

Parent topic:*Run a demo using MCUXpresso IDE*

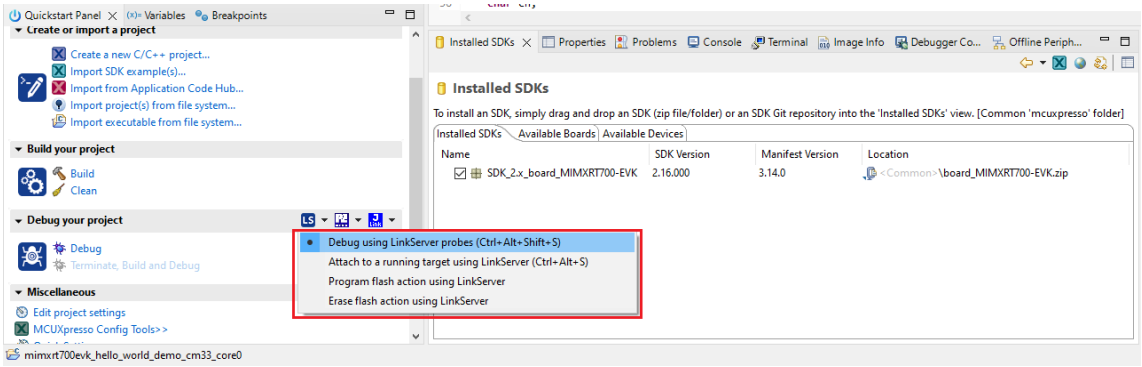
Run an example application To download and run the application, perform the following steps:

1. Determine the debug interface that comes loaded on your specific hardware platform.
   - For MIMXRT700-EVK, MCU-LINK is used.
   - J54 is used as the debugging port which provides both debugging functionality and console out.
2. Connect USB cable between J54 on EVK and PC USB port.
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:
   - 115200 baud rate or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
   - No parity
   - 8 data bits

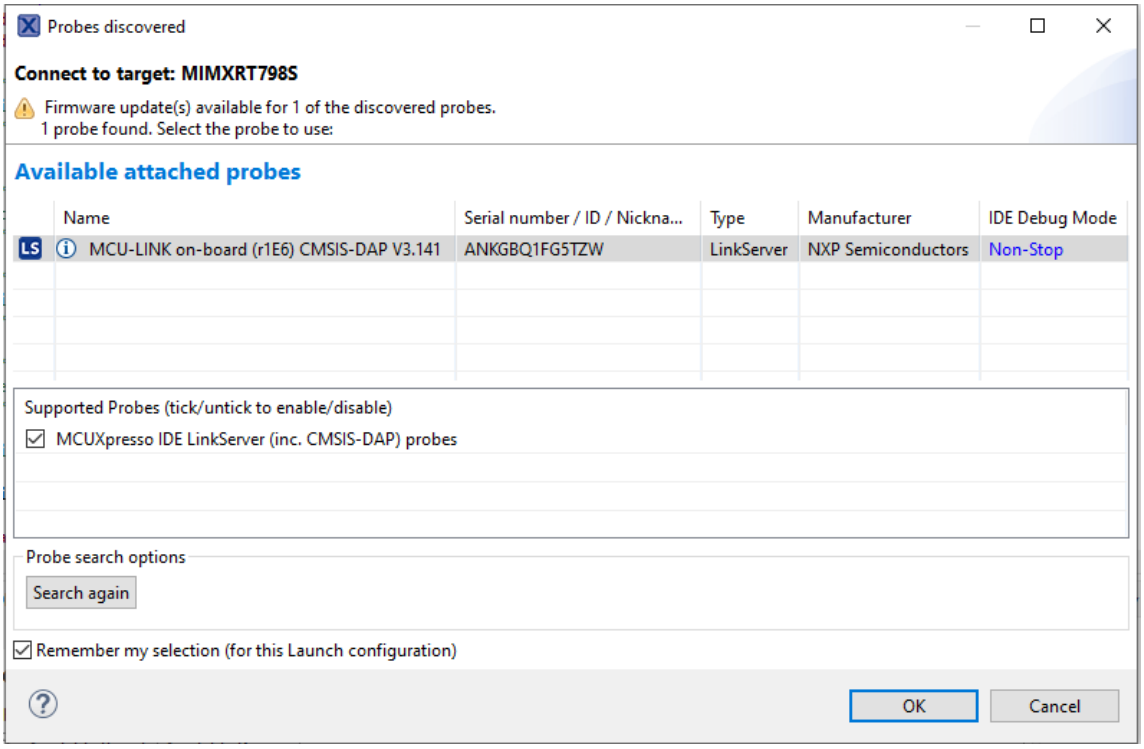


- 1 stop bit

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

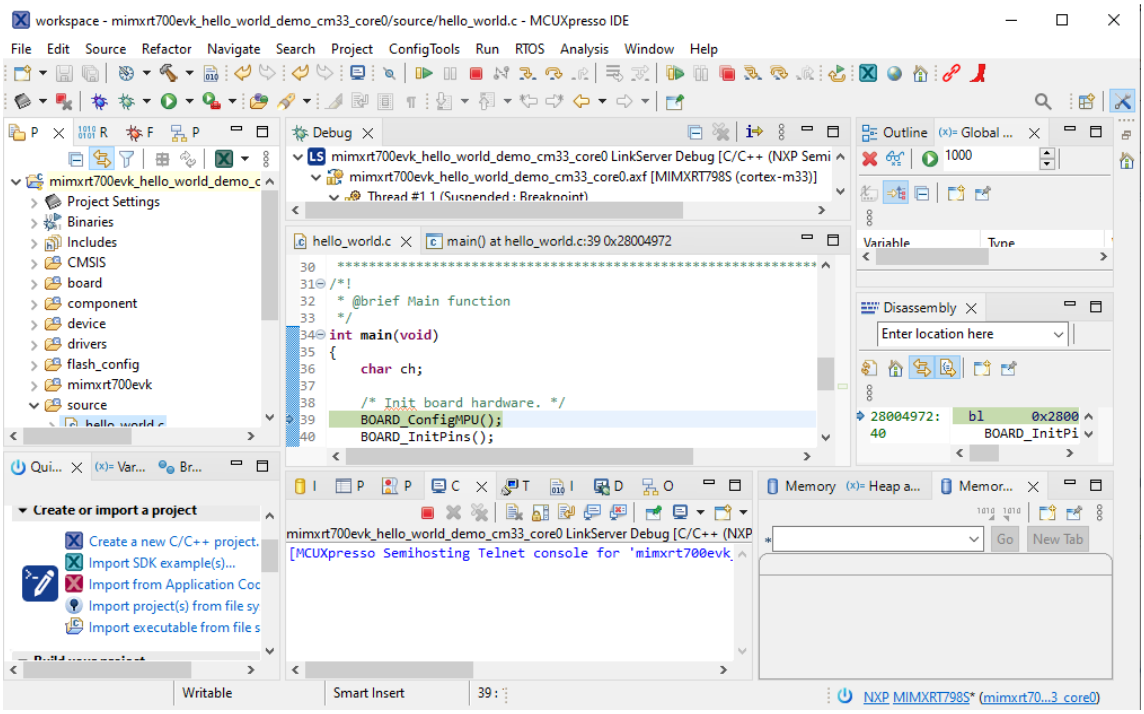


5. The first time you debug a project, the **Debug Emulator Selection** dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)



Note: Make sure that the board is set to XSPI0 flash boot mode.

6. The application is downloaded to the target and automatically runs to main().

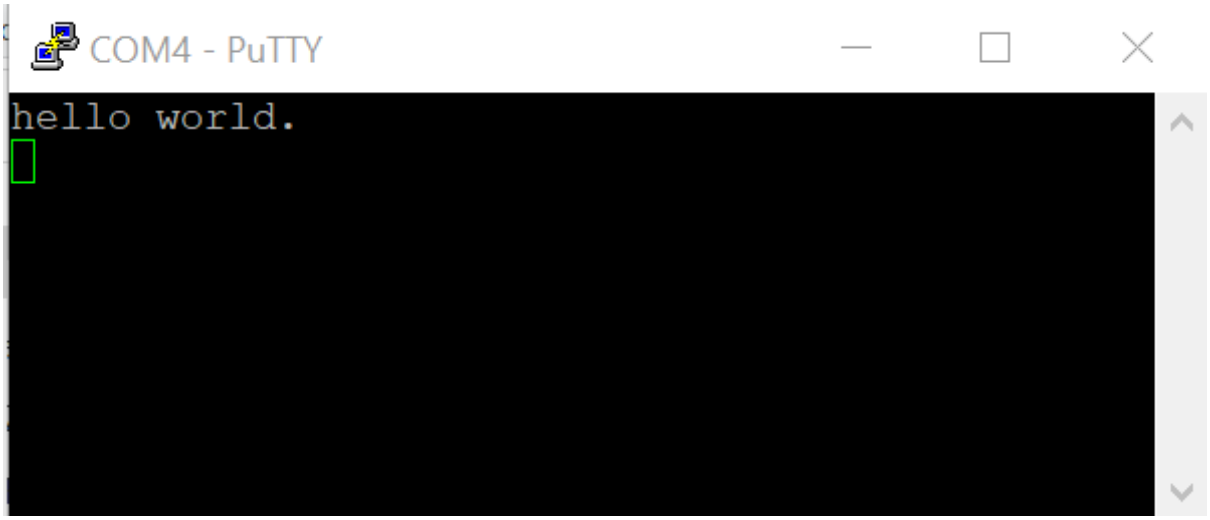


7. To start the application, click the **Resume**.



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

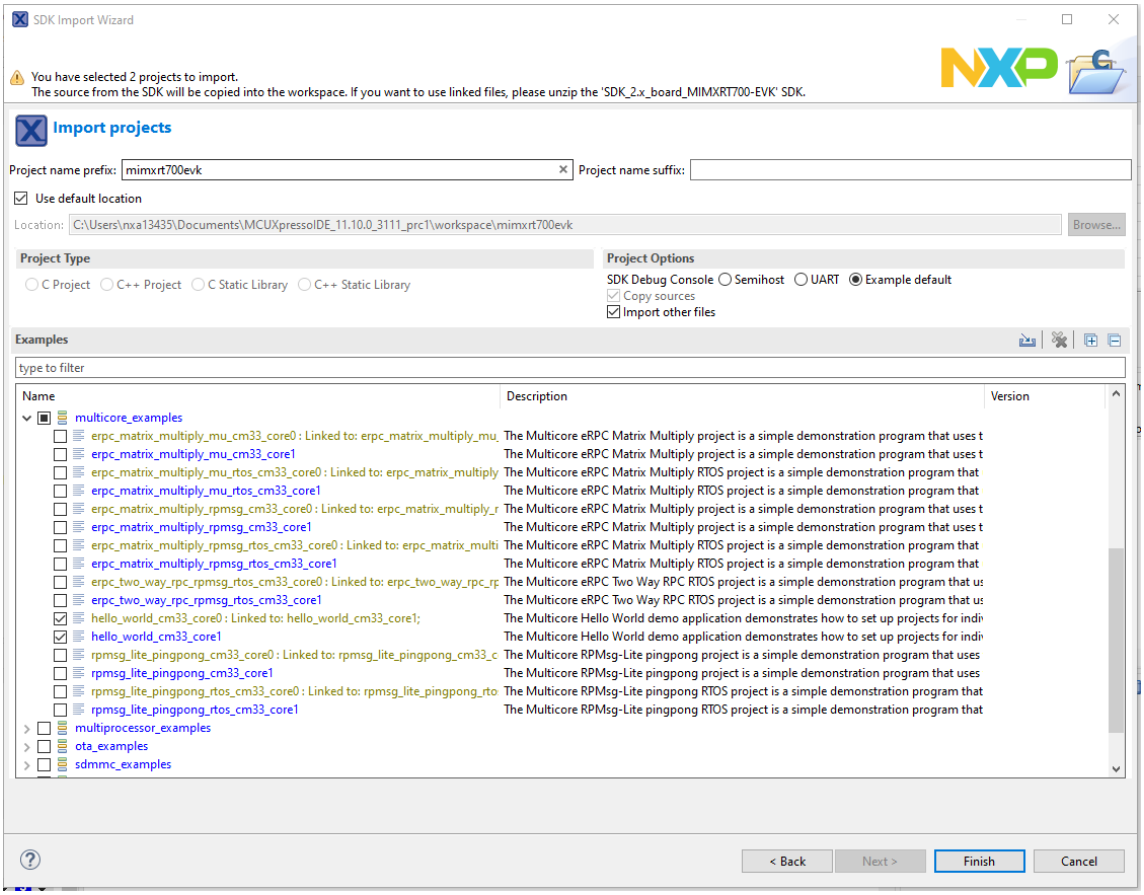
check your terminal settings and connections.



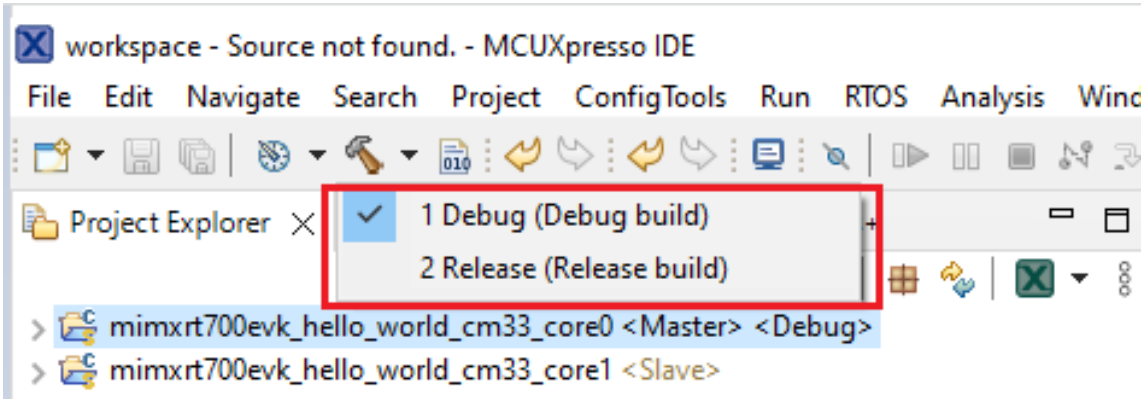
Parent topic:*Run a demo using MCUXpresso IDE*

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 **mimxrt700evk** 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 is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, select **mimxrt700evk** and click **Next**.
2. Expand the *multicore_examples* folder and select **hello_world_cm33_core0**. The hello_world_cm33_core1 counterpart project is automatically imported with the cm33_core0 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 hello_world_cm33_core0 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 *Figure 2*. For this example, select **Debug**.

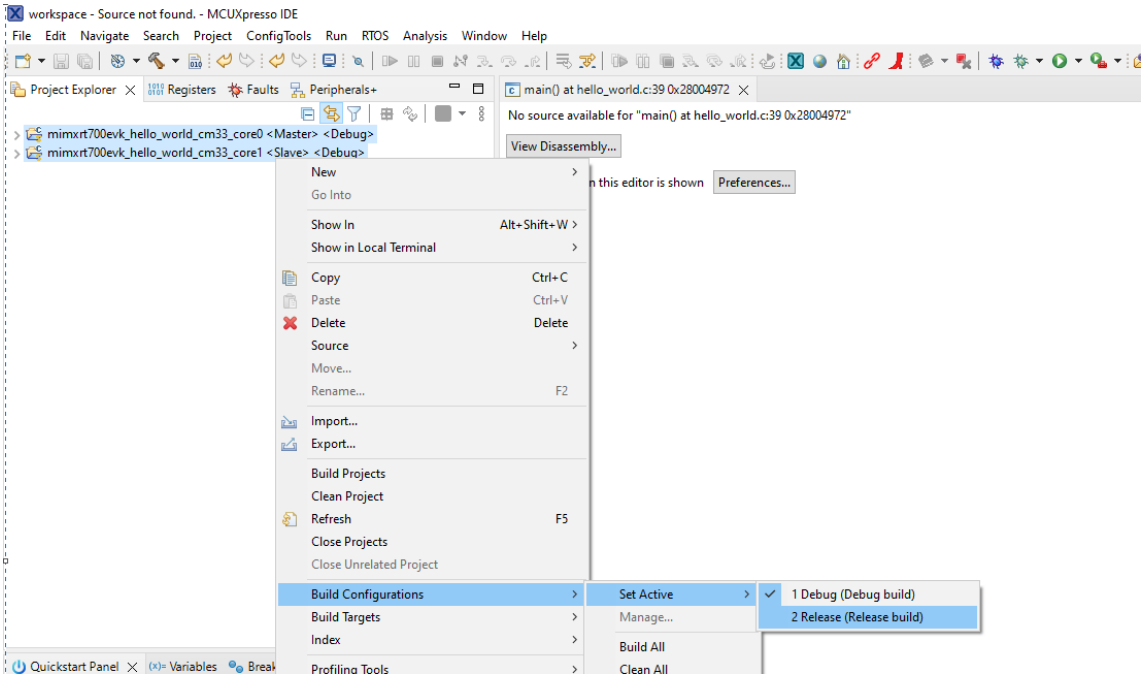


Press the **Build** button to start the multi-core project build.. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm33_core0) also makes the referenced auxiliary core application (cm33_core1) to build.

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

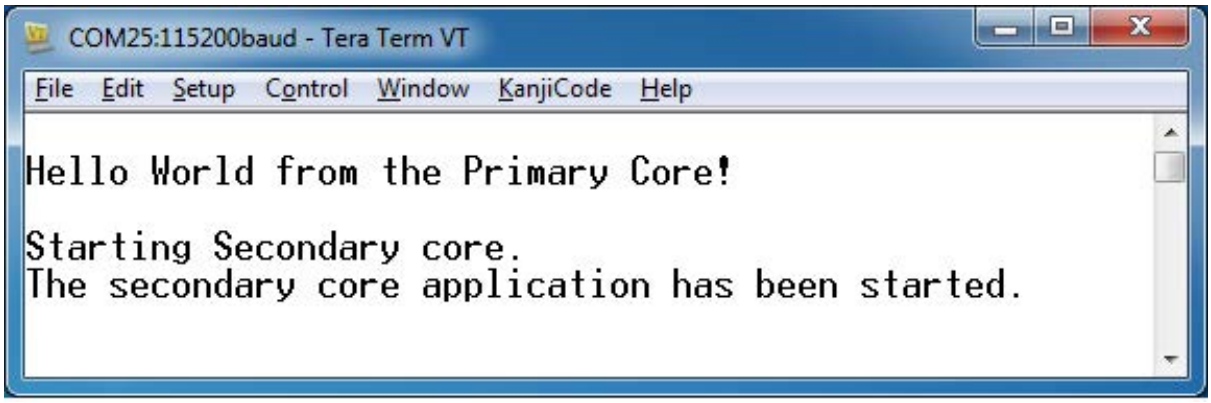
triggering the primary core application (cm33_core0) build.



Parent topic:*Run a demo using MCUXpresso IDE*

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. Select the cm33_core1 project and start debugging the cm33_core1 project. Then, select the cm33_core0 project and start debugging the cm33_core0 project.

After clicking **Resume All Debug** sessions, 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.



Parent topic:*Run a demo using MCUXpresso IDE*

Run a demo application using IAR

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

Note:

IAR Embedded Workbench for Arm version 9.50.1 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 for MIMXRT700-EVK* (document MCUXSDKRT798RN).

When debug or run the flash_debug or flash_release targets, make sure that the board is configured as XSPI0 boot mode (SW10 1 - OFF, 2 - ON).

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

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

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

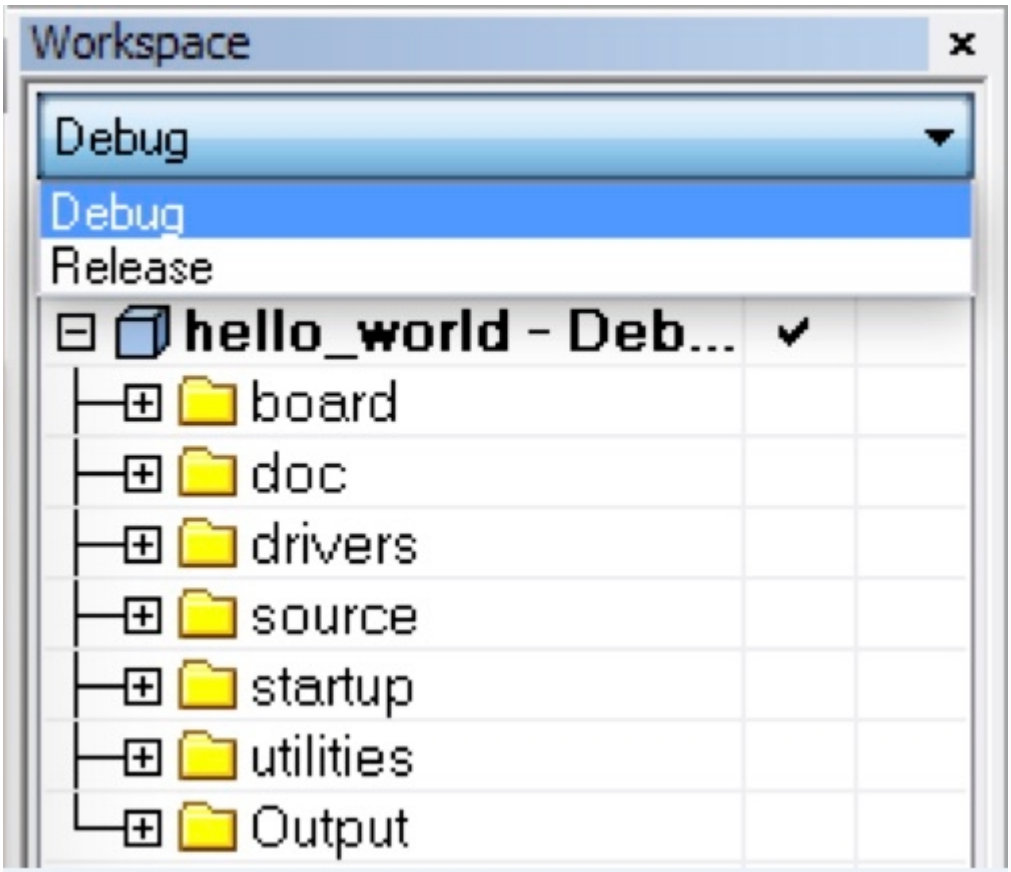
   Using the MIMXRT700-EVK hardware platform as an example, the hello_world workspace is located in:

```
<install_dir>/boards/mimxrt700evk/demo_apps/hello_world/cm33_core0/iar/hello_world_cm33_
↪core0.eww
```

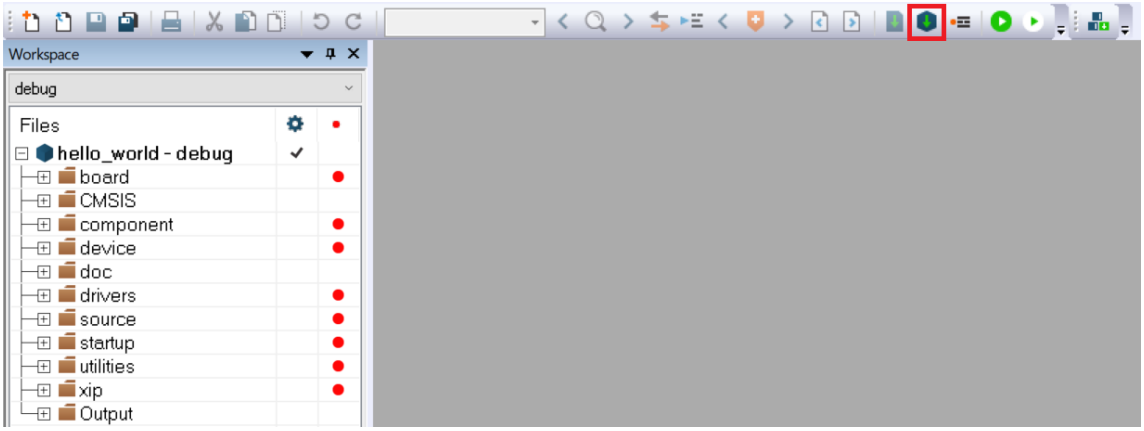
   Other example applications may have additional folders in their path.

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

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



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

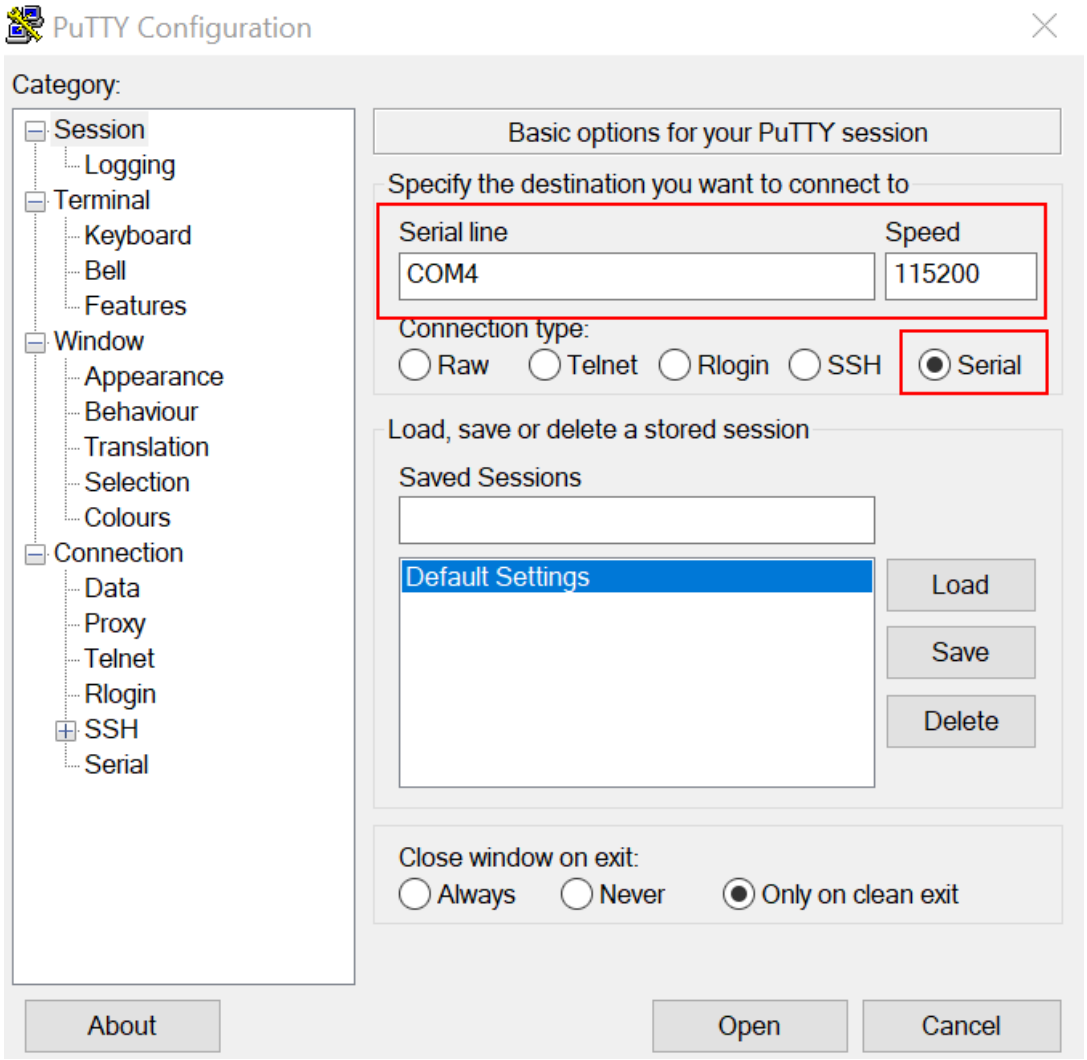


4. The build completes without errors.

Parent topic:*Run a demo application using IAR*

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

1. To get the MCU-Link firmware ready, see *Updating debugger firmware*.
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

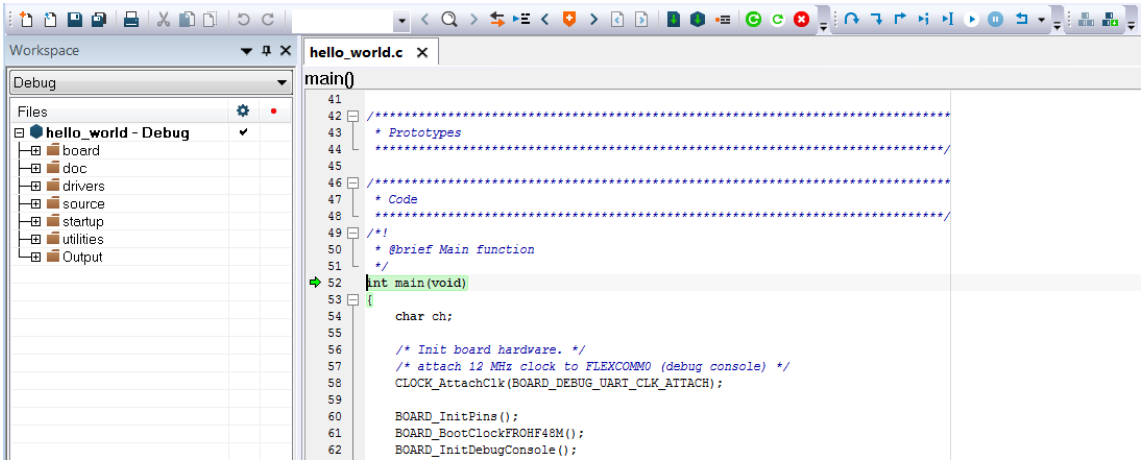


4. 1 stop bit

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



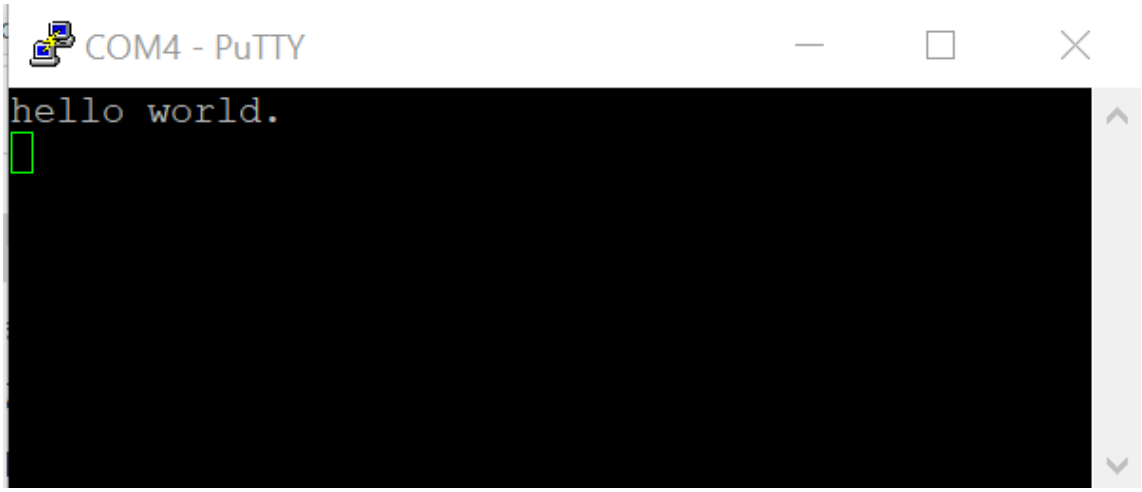
5. The application is then downloaded to the target and automatically runs to the main() function.



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.



Parent topic:*Run a demo application using IAR*

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

<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_id>/iar

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

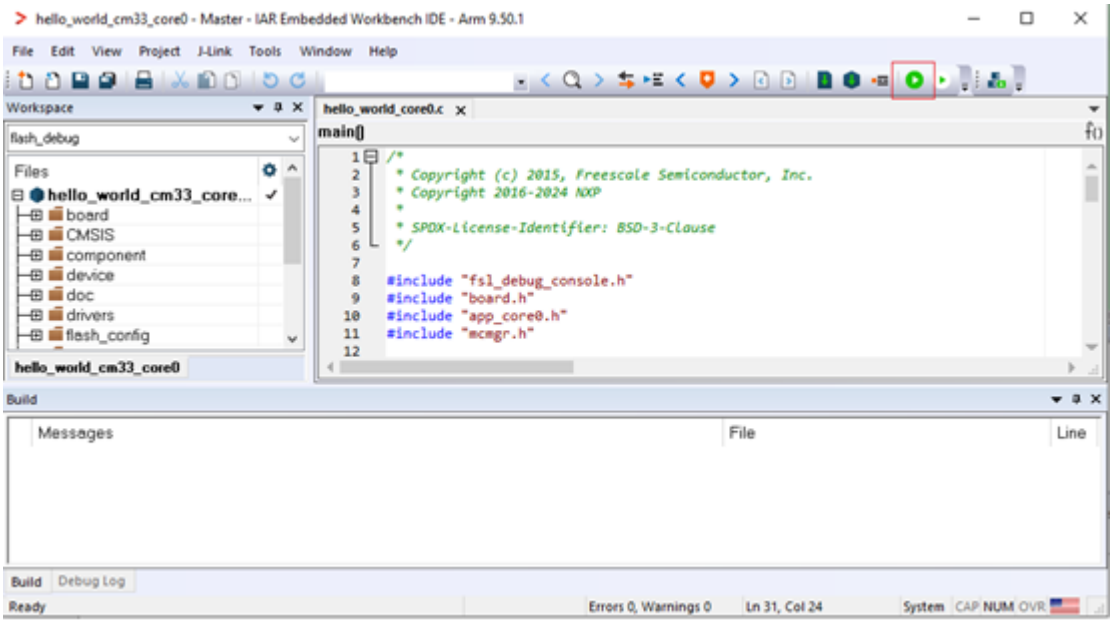
<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core0/iar/hello_world_cm33_core0.ew

<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core1/iar/hello_world_cm33_core1.ew

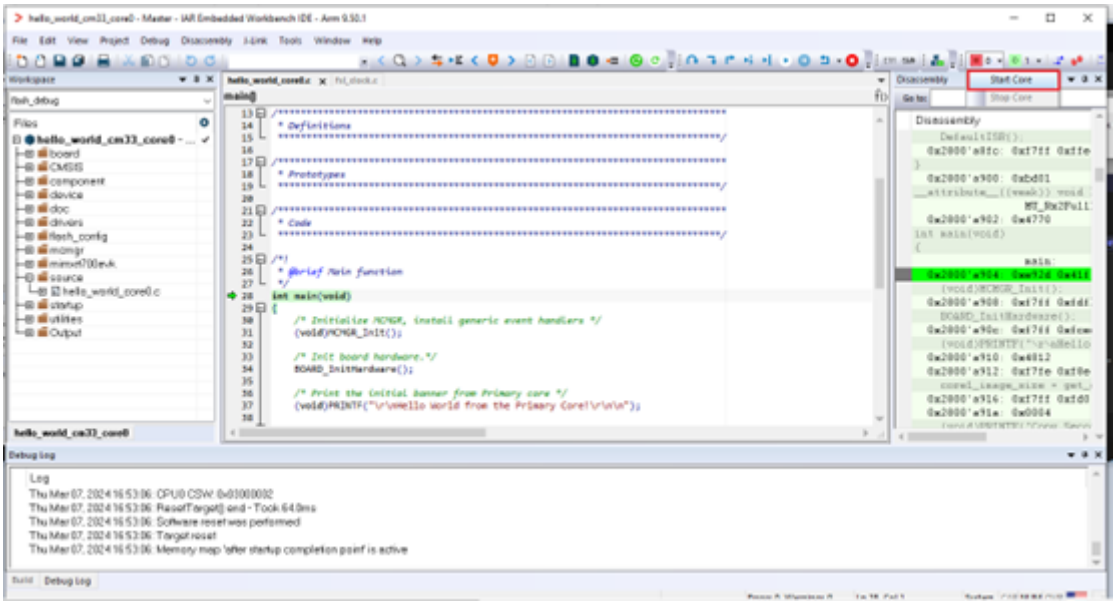
Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm33_core1) first, because the primary core application project (cm33_core0) must know the auxiliary core application binary when running the linker. When the auxiliary core application binary is not ready, it is impossible to finish the primary core linker.

By default, the primary core flash_debug target links the auxiliary core debug target, and the primary core flash_release target links the auxiliary core release target. During the primary core execution, the auxiliary core image is copied from flash into the SRAM and executed.

1. Build the CM33_core1 and CM33_core0 projects respectively.
2. Only click the **Download** and **Debug** button on the CM33_core0 project, IAR could help start to debug a multicore project. It is user-friendly to debug multicore examples with CMSIS-DAP on IAR (Multicore Project is set on **Debugger** → **Multicore** window).



3. Start core0 and then start core1 on the **CM33_core0** project.



4. The Hello_World multicore demos are now running. A banner appears on the terminal and the LED blinks. If this is not true, check your terminal settings and connections.

```
Hello World from the Primary Core!

Copy Secondary core image to address: 0x20600000, size: 12272
Starting Secondary core.
The secondary core application has been started.
```

Parent topic:*Run a demo application using IAR*

Run a demo using Keil MDK/µVision

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

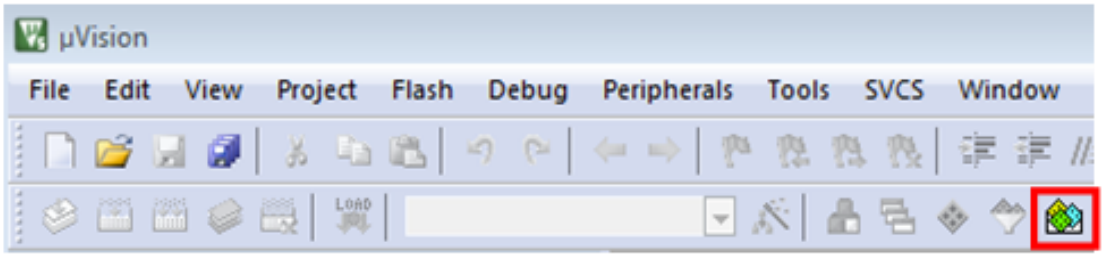
The hello_world demo application is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

Note:

Make sure that the board is configured as XSPI0 boot mode when debugging or running the examples.

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.

Parent topic:*Run a demo using Keil MDK/µVision*

Build an example application To build an example application, perform the following steps:

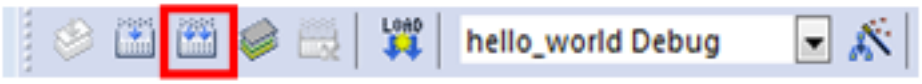
1. Open the desired example application workspace in:

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/<core_id>/mdk
   The workspace file is named as <demo_name>.uvmpw.

   For this specific example, the actual path is:

   <install_dir>/boards/mimxrt700evk/demo_apps/hello_world/cm33_core0/mdk/hello_world_demo_cm33_core

2. To build the demo project, select **Rebuild**, as shown in *Figure 1*, highlighted in red.



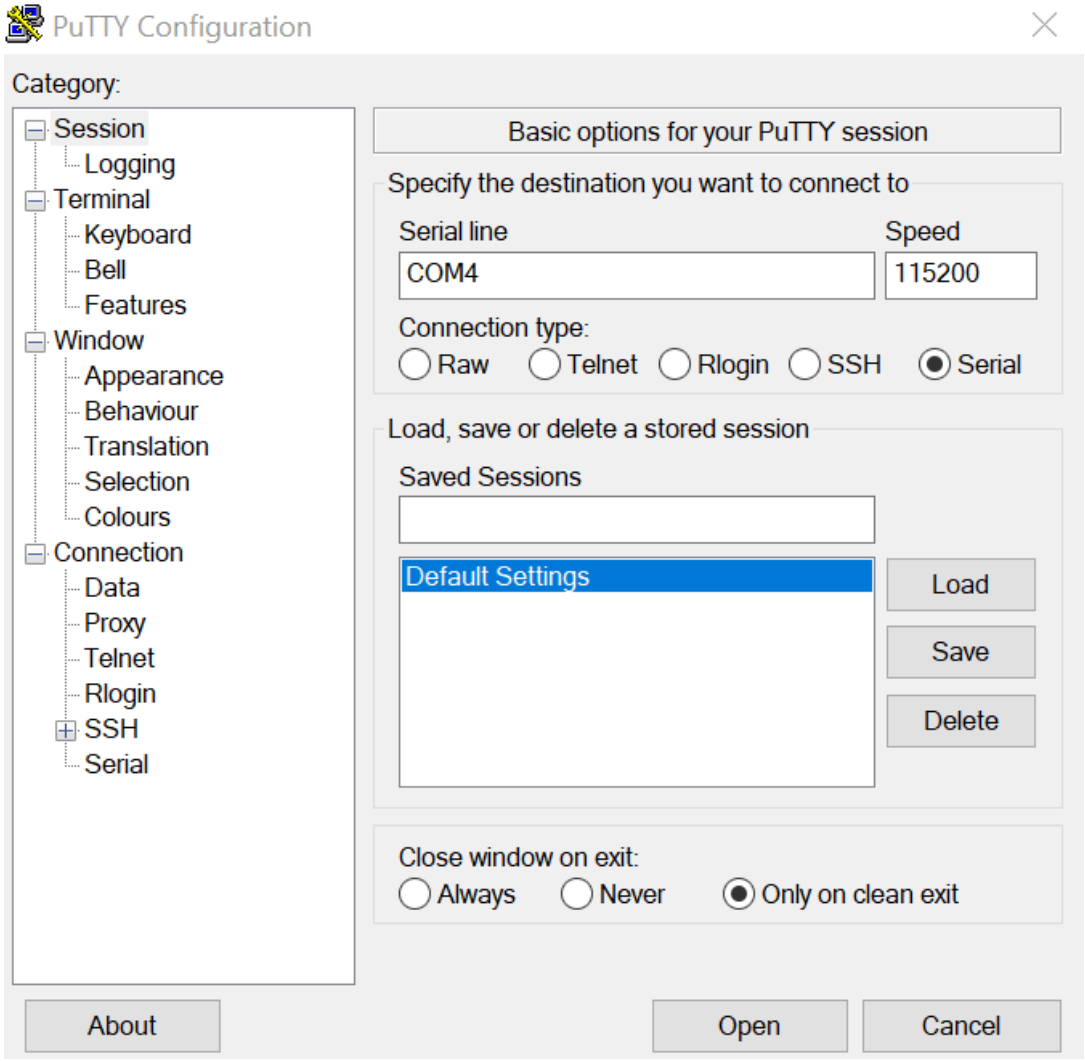
3. The build completes without errors.

Parent topic:*Run a demo using Keil MDK/µVision*

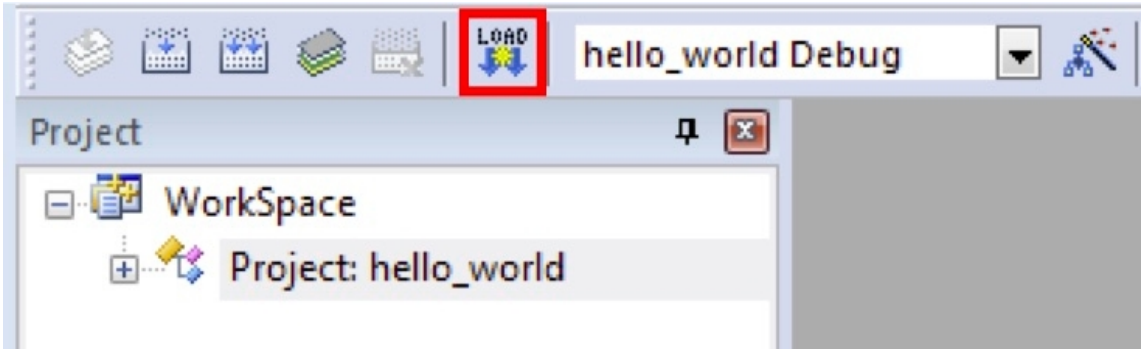
Run an example application To download and run the application, perform these steps:

1. Determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with the CMSIS-DAP/mbed/DAPLink interface, visit Windows serial configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux operating system, this step is not required.
   - If using J-Link either a standalone debug pod or MCU-LINK, install the J-Link software (drivers and utilities) from Segger.
2. Connect the development platform to your PC via USB cable using MCU-LINK 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:
   - 115200 or 9600 baud rate, depending on your board (reference the BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
   - No parity
   - 8 data bits
   - 1 stop bit

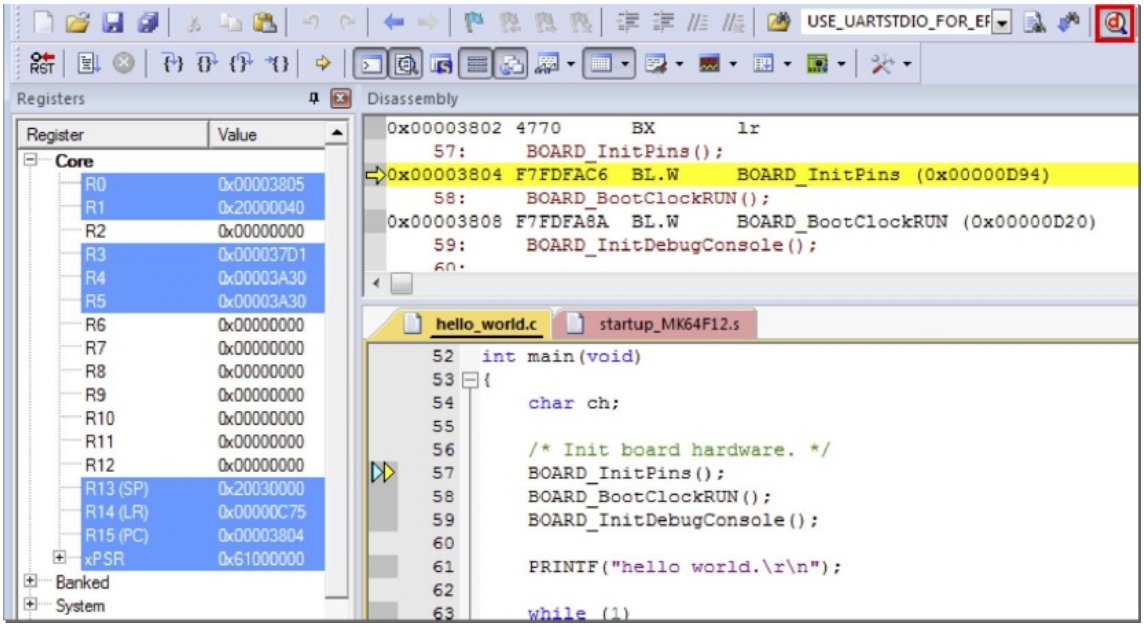


4. In µVision, after the application is built, click the **Download** button to download the application to the target.

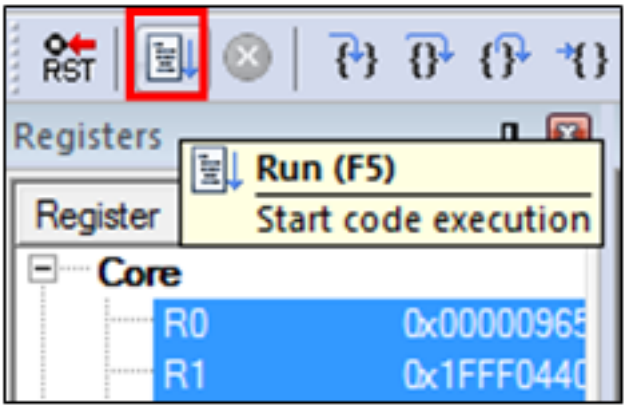


5. After clicking the **Download** button, the application downloads to the target and is running.

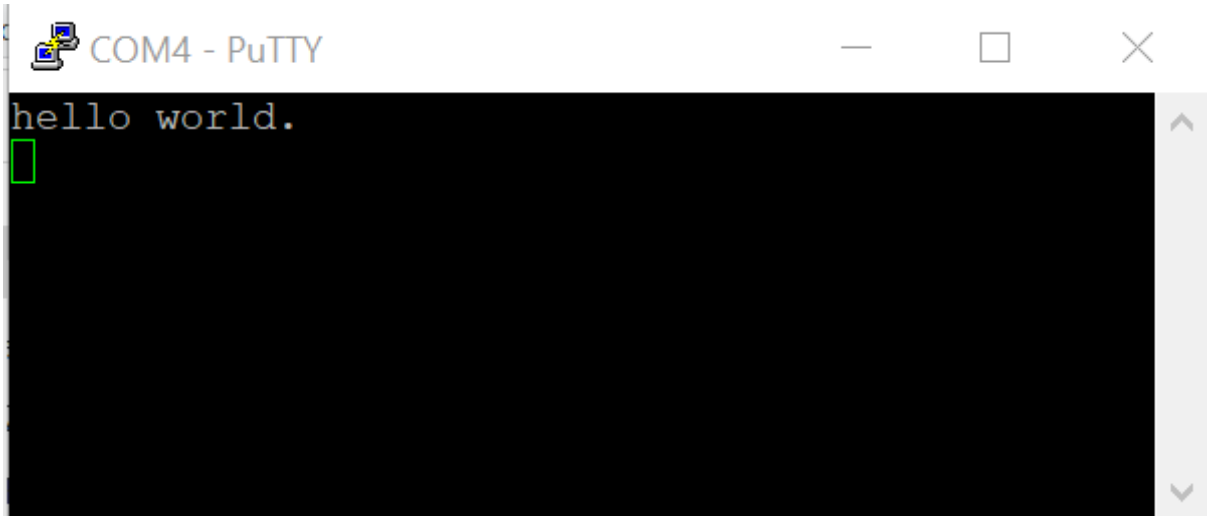
To debug the application, click the **Start/Stop Debug Session** button, highlighted in red in *Figure 3*.



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.



Parent topic:*Run a demo using Keil MDK/µVision*

Build a multicore example application This section describes the particular steps that need to be done in order to build and run a multicore application.

The multicore Hello World MDK workspaces are located in this folder:

<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core0/mdk/hello_world_cm33_core0.u

<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core1/mdk/hello_world_cm33_core1.u

To build both applications separately, click the **Rebuild** button. Build the application for the auxiliary core (cm33_core1) first, because the primary core application project (cm33_core0) 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.

Parent topic:*Run a demo using Keil MDK/µVision*

Run a multicore example application This section describes the steps to run a multicore example application. The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory.

1. To download and run the multicore application, switch to the primary core application project and perform Step 1 – Step 3, as described in Run an example application.
2. Run the primary core project and then run the secondary core project.
3. Hello_World multicore demos are now running. A banner appears on the terminal and the LED blinks. If this is not true, check your terminal settings and connections.

```
Hello World from the Primary Core!

Copy Secondary core image to address: 0x20600000, size: 12844
Starting Secondary core.
The secondary core application has been started.
```

Parent topic:*Run a demo using Keil MDK/µVision*

Run a demo using Arm GCC

This section describes the steps to configure the command-line Arm GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The hello_world demo application is targeted for the MIMXRT700-EVK hardware platform which is used as an example.

Note:

ARMGCC version 12.3.Rel1 is used as an example in this document. The latest GCC version for this package is as described in the *MCUXpresso SDK Release Notes for MIMXRT700-EVK* (document MCUXSDKRT798RN).

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

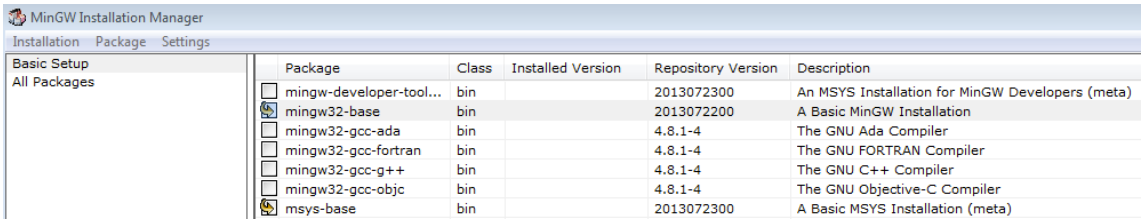
Install GCC Arm Embedded tool chain Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes for MIMXRT700-EVK* (document MCUXSDKRT798RN).

Parent topic: Set up toolchain

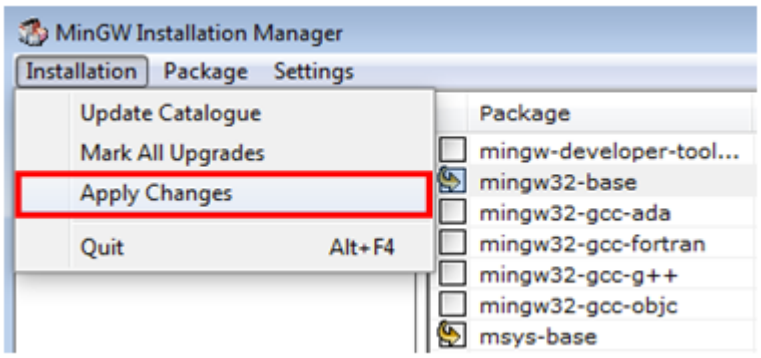
Install MinGW (only required on Windows operating system) The Minimalist GNU for Windows (MinGW) development tools provide a set of tools that are not dependent on third-party C-Runtime DLLs (such as Cygwin). The build environment used by the MCUXpresso SDK does not use the MinGW build tools, but does leverage the base install of both MinGW and MSYS. MSYS provides a basic shell with a Unix-like interface and tools.

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

   Note: The installation path cannot contain any spaces.
3. Ensure that the **mingw32-base** and **msys-base** are selected under **Basic Setup**.



4. In the **Installation** menu, click **Apply Changes** and follow the remaining instructions to complete the installation.

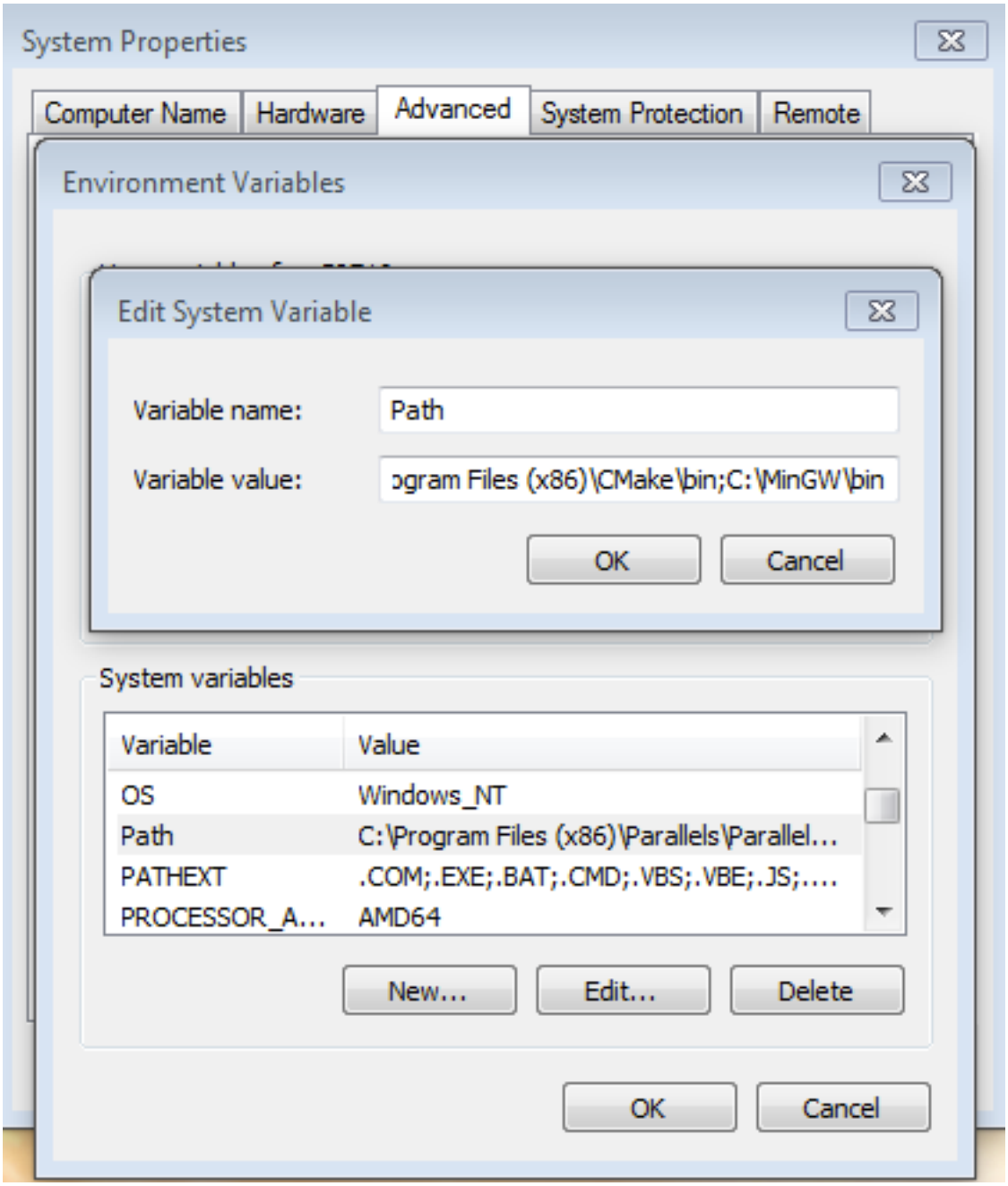


5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel** > **System and Security System** > **Advanced System Settings** in the **Environment Variables...** section. The path is:

```
<mingw_install_dir>\bin
```

   Assuming the default installation path, C:\MinGW, an example is shown below. If the path is not set correctly, the toolchain will not work.

   Note: If you have C:\MinGW\msys\x.x\bin in your PATH variable (as required by Kinetis SDK 1.0.0), remove it to ensure that the new GCC build system works correctly.



Parent topic: Set up toolchain

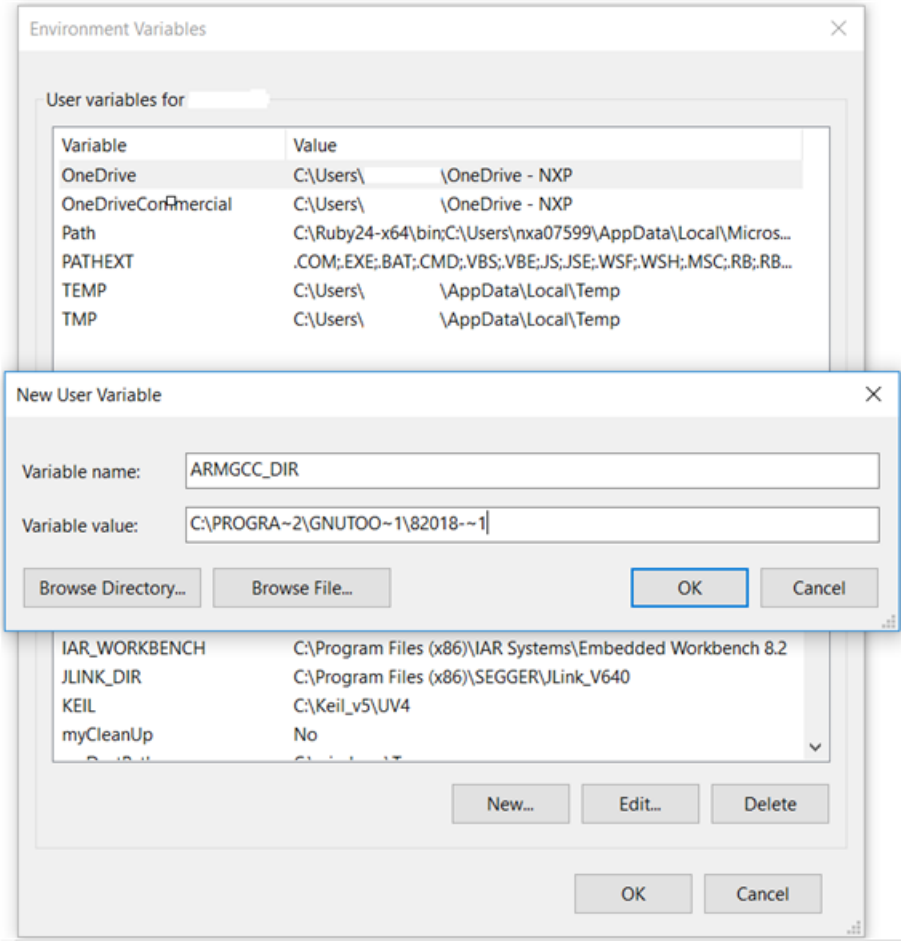
Add a new system environment variable for ARMGCC_DIR Create a new *system* environment variable and name it as ARMGCC_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

See the installation folder of the GNU Arm GCC Embedded tools for the exact pathname of your installation.

Short path should be used for path setting, you could convert the path to short path by running command for %I in (.) do echo %~sI in above path.

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

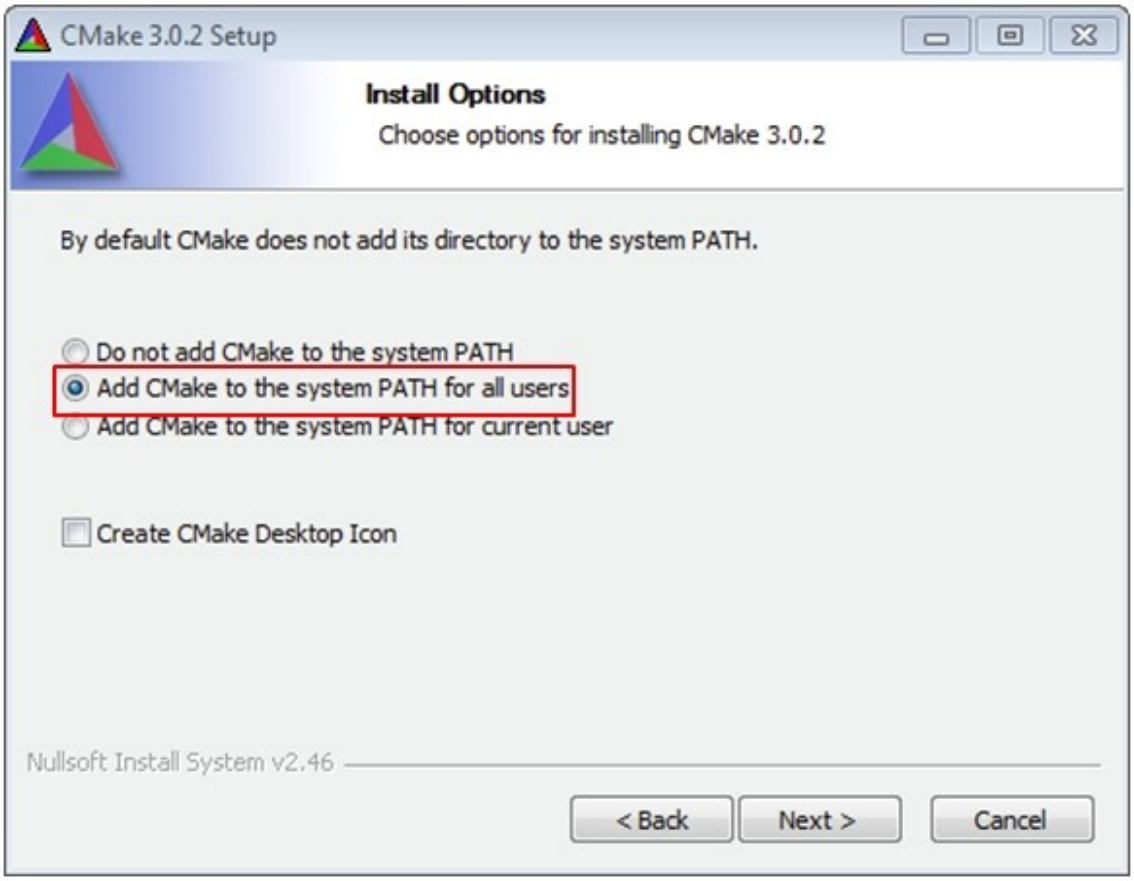
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018-~1
C:\PROGRA~2\GNUTOO~1\82018-~1
```



Parent topic: Set up toolchain

Install CMake

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



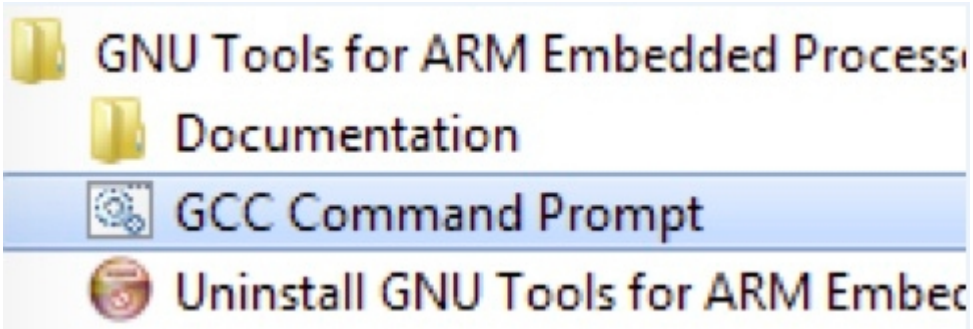
3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.
5. Make sure sh.exe is not in the Environment Variable PATH. This is a limitation of mingw32-make.

Parent topic: Set up toolchain

Parent topic:*Run a demo using Arm GCC*

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

1. Open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system **Start** menu, go to **Programs** >**GNU Tools Arm Embedded <version>** and select **GCC Command Prompt**.



2. Change the directory to the example application project directory which has a path similar to the following:

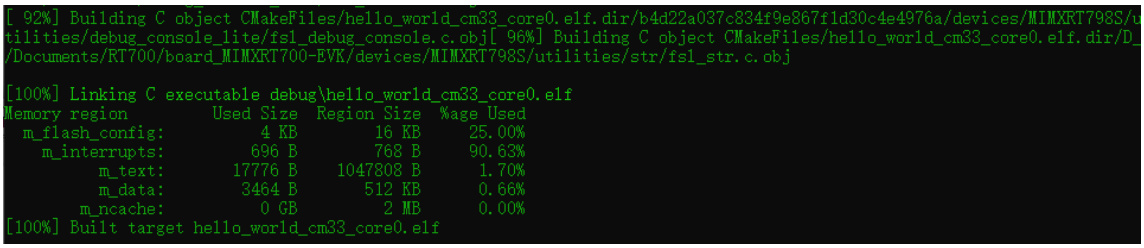
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc
```

For this example, the exact path is:

```
<install_dir>/boards/mimxrt700evk/demo_apps/hello_world/cm33_core0/armgcc
```

Note: To change directories, use the cd command.

3. Type **build_debug.bat** on the command line or double click on **build_debug.bat** file in Windows Explorer to build it. The output is as shown in *Figure 2*.



Parent topic:*Run a demo using Arm GCC*

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

<install_dir>/boards/mimxrt700evk/multicore_examples/<application_name>/<core_id>/armgcc

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

<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core0/armgcc/build_debug.bat
<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33/armgcc_core1/build_flash_debug.bat

Build the application for the auxiliary core (cm33_core1) first, because the primary core application project (cm33_core0) 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.

By default, the primary core flash_debug target links the auxiliary core **debug** target, and the primary core flash_release target links the auxiliary core **release** target. During the primary core execution, the auxiliary core image is copied from flash into SRAM and executed.

Parent topic:*Run a demo using Arm GCC*

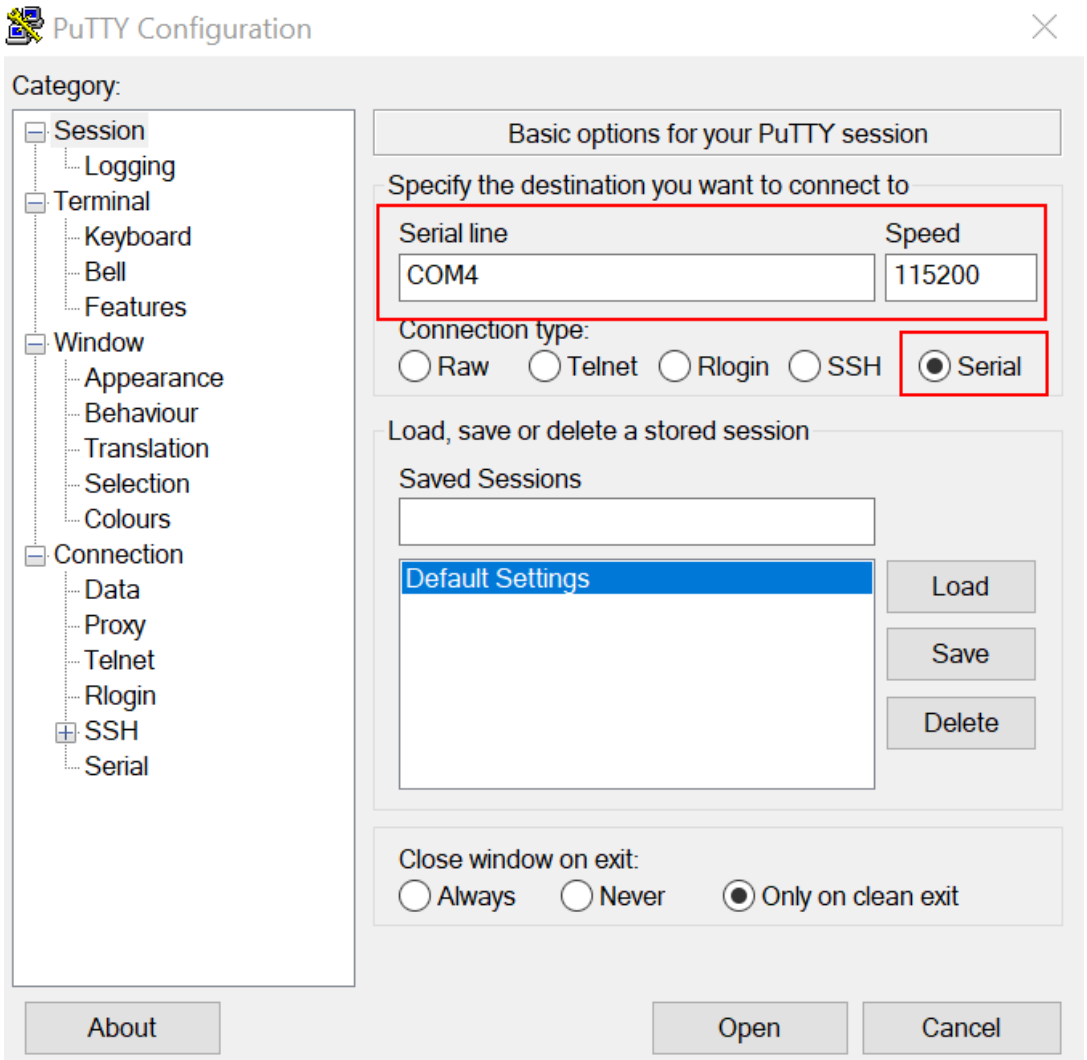
Run an example application This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, make sure that either:

- The MCU-Link firmware on your board is programmed with the J-Link MCU-Link firmware. For instructions on reprogramming the MCU-Link firmware, see *Updating debugger firmware*. If your board does not support MCU-Link, a standalone J-Link pod is required.
- You have a standalone J-Link pod that is connected to the debug interface of your board. Note that some hardware platforms require hardware modification in order to function correctly with an external debug interface.

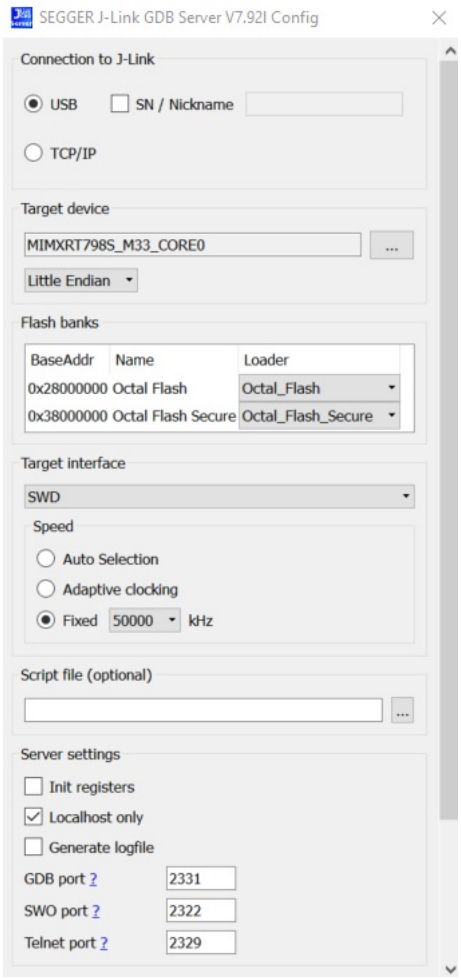
After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the development platform to your computer via USB cable between the MCU-Link USB connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.

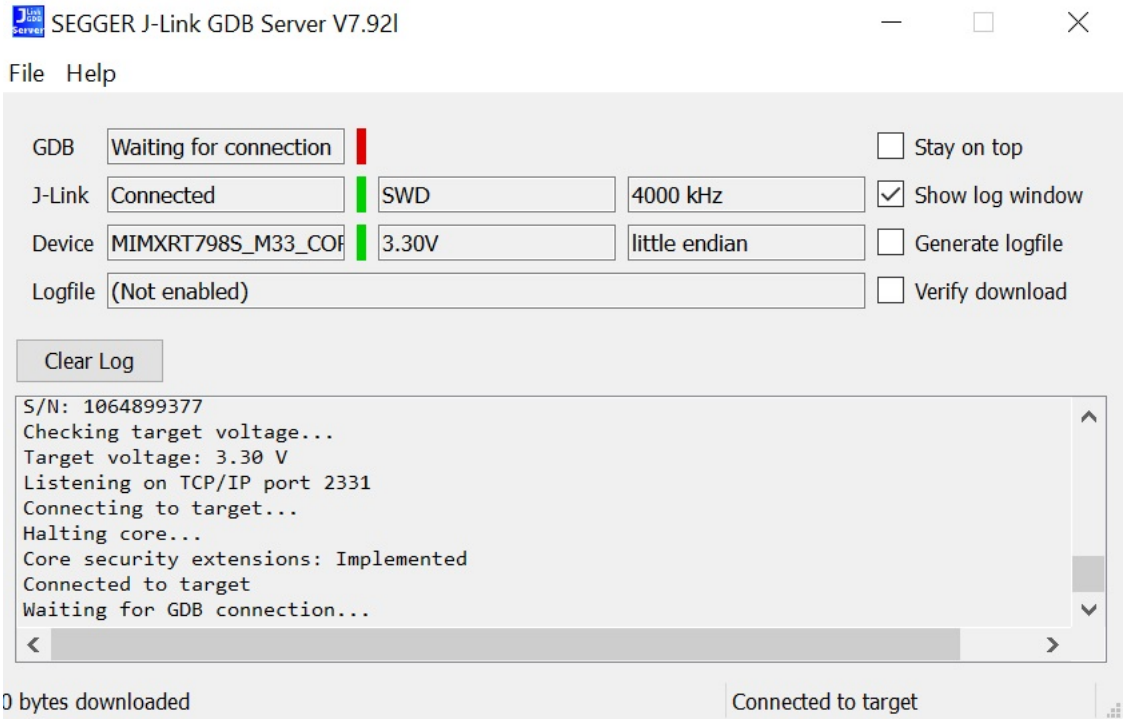
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
    1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
    2. No parity
    3. 8 data bits
    4. 1 stop bit



3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system **Start** menu and selecting **Programs->SEGGER->J-Link <version> J-Link GDB Server**.
4. Modify the settings as shown in *Figure 1*. The target device selection chosen for this example is **MIMXRT798S_M33_CORE0**.

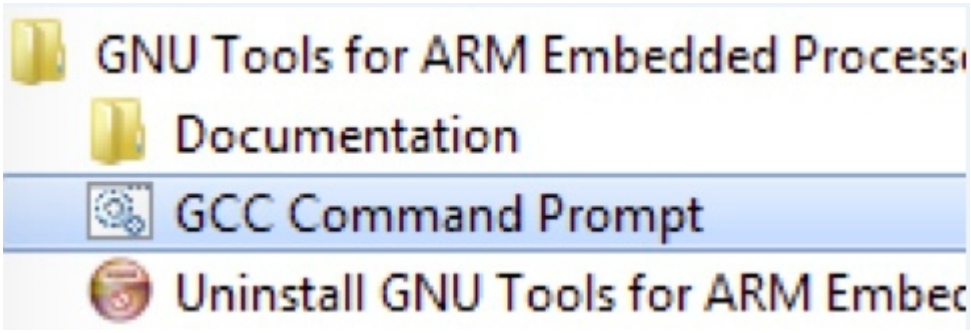


5. After it is connected, the screen should be as shown in *Figure 3*.



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

Tools Arm Embedded <version> and select **GCC Command Prompt**.



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

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

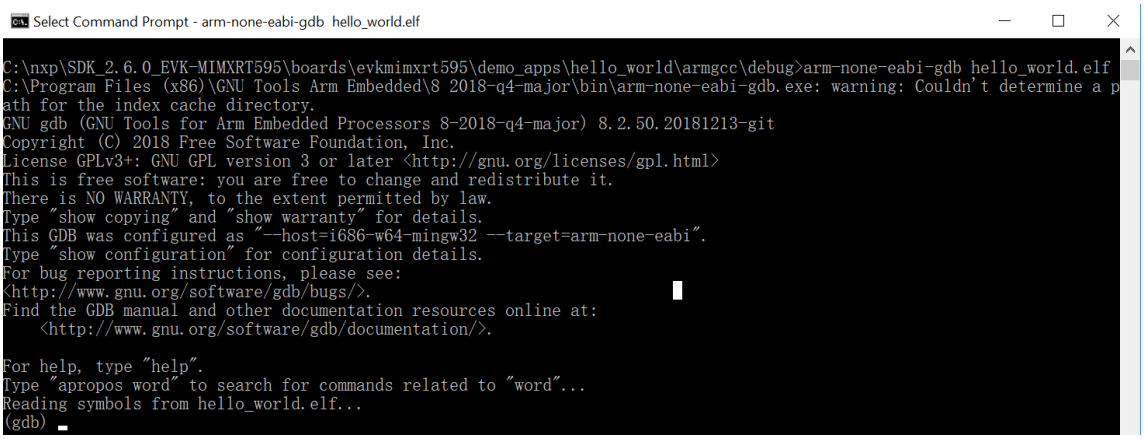
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/release
```

   For this example, the path is:

```
<install_dir>/boards/mimxrt700evk/demo_apps/hello_world/cm33_core0/armgcc/debug
```

```
<install_dir>/boards/mimxrt700evk/demo_apps/hello_world/cm33_core0/armgcc/release
```

8. Run the arm-none-eabi-gdb.exe <application_name>.elf command. For this example, it is arm-none-eabi-gdb.exe hello_world_cm33_core0.

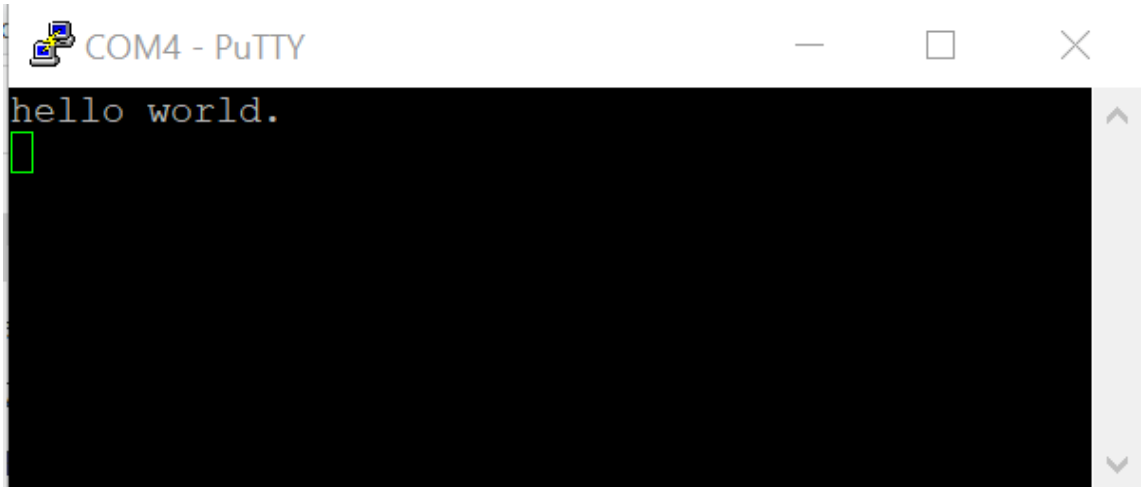


   Note: Make sure that the board is set to XSPI flash boot mode before debugging.

9. Run these commands:
   1. target remote localhost:2331
   2. monitor reset
   3. monitor halt
   4. load
   5. monitor reset

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

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



Parent topic:*Run a demo using Arm GCC*

Build and run the ezhv_examples

This section describes the steps required to configure the environment to build, run, and debug ezhv example applications. The *<install_dir>/boards/<board_name>/ezhv_examples/led_blinky* is used as an example.

For each example, generally there are two folders, one for the cm33_core0 core and the other for the EZHV core. For the led_blinky example, they are:

<install_dir>/boards/<board_name>/ezhv_examples/led_blinky/cm33_core0

<install_dir>/boards/<board_name>/ezhv_examples/led_blinky/ezhv

Build the application for the auxiliary core (ezhv) first, because the primary core application project (cm33_core0) must know the auxiliary core application binary when running the linker. When the auxiliary core application binary is not ready, it is impossible to finish the primary core linker. This section focuses on the auxiliary core (ezhv). For details about building the primary core for different toolchain, see the corresponding chapter.

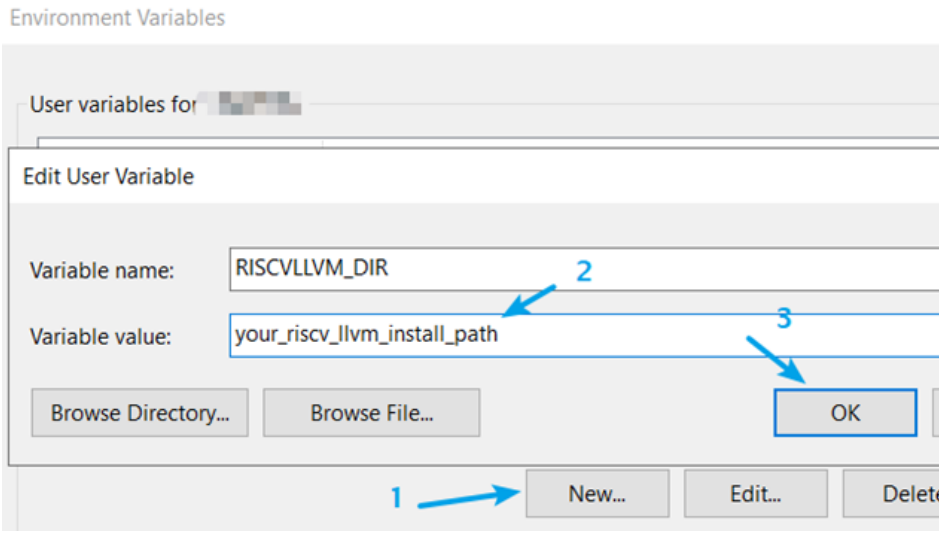
Set the toolchain for EZHV To set the toolchain for EZHV, perform the following steps:

1. Download and install the RISCV LLVM toolchain.
2. Add a system environment variable RISCVLLVM_DIR, and set the value to the RISCV LLVM install path.
   - For Linux or Mac, run the command:

     ```
     export RISCVLLVM_DIR=<your_risc_llvm_install_path>
     ```

   - For Windows, as shown in Set the toolchain for EZHV, perform the following steps:
     1. Click **Windows Start** -> **Settings**.
     2. Search and open **Edit environment variable for your account**

3. Create a new variable.



3. Download Ninja from this link. Add the path ninja executable to system environmental variable **PATH**. When it finishes, run the ninja – version in command line to verify the installation. The result looks like:

```
$ ninja --version
1.10.1
```

4. Download and install CMake.

Parent topic:*Build and run the ezhv_examples*

Build the EZHV project To build the EZHV project, go to the project folder *<install_dir>/boards/<board_name>/ezhv_examples/<example>/ezhv/riscvllvm*.

- For Windows, click the **build_debug.bat**.
- For Linux or Mac, run the build_debug.sh in the command line.

When the build finishes, a binary *ezhv_image.bin* is generated in the folder *<install_dir>/boards/<board_name>/ezhv_examples/<example>/ezhv/binary*.

Parent topic:*Build and run the ezhv_examples*

Run the project There are two ways to run the project: Run with primary core, or run independently.

- Run with primary core.

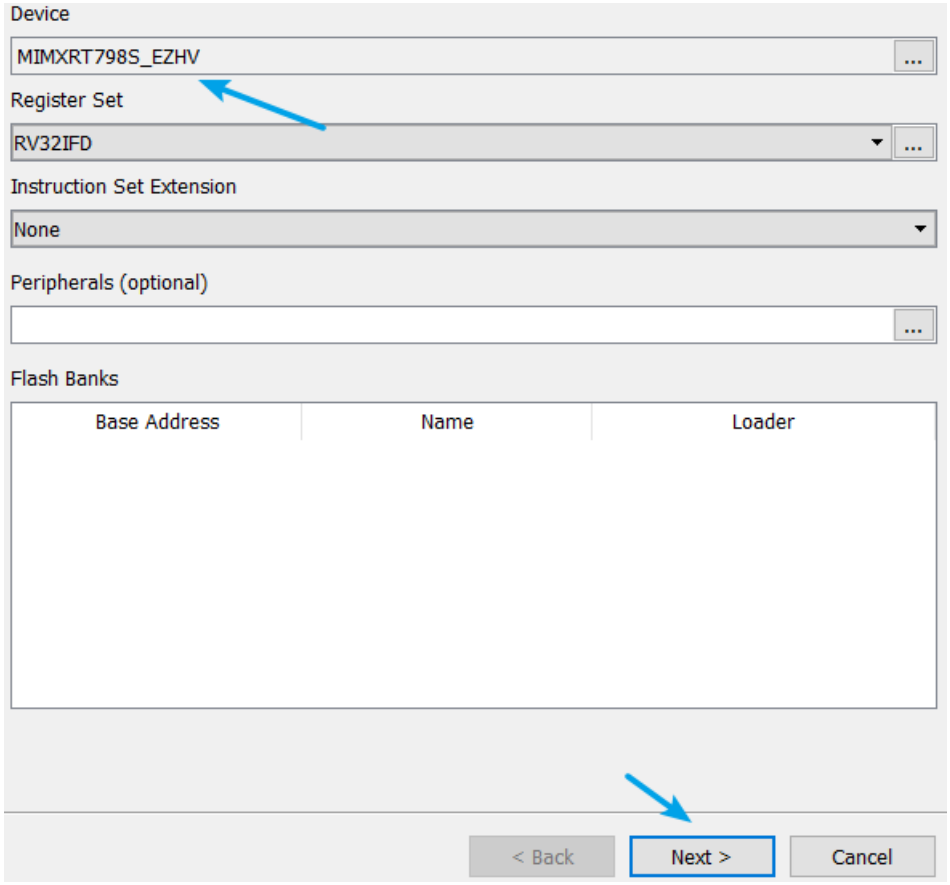
  After EZHV binary *ezhv_image.bin* is created, follow the primary core project guide to build the primary core project and run.

- Debug the EZHV independently.

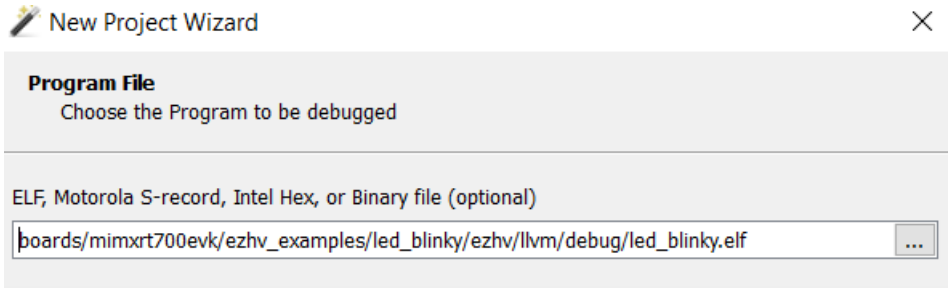
  Currently SEGGER Ozone can be used to run and debug the EZHV executable.

  1. An external J-Link plus is needed, connect JP18 1-2 on the mimxrt700evk board to use external debugger and connect external J-Link to J18.
  2. Download and install the J-Link.
  3. Download and install the SEGGER Ozone.
  4. Open Ozone and create project, click **File** -> **New** -> **New Project Wizard**.

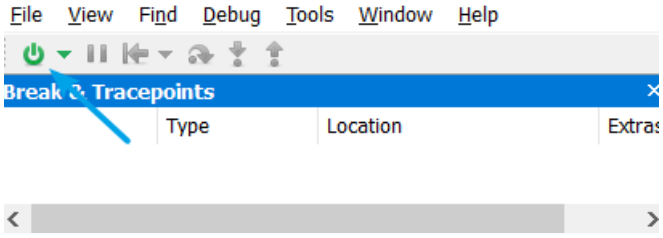
5. Select the device **MIMXRT798S_EZHV**.



6. Select the built EZHV executable.



7. On the Ozone window, click the button to run.



Parent topic:*Build and run the ezhv_examples*

How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

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 Compute Domain and the other is for Sense Domain.

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.
3. 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.

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
}
```

Updating debugger firmware

Updating MCU-Link board firmware MCU-Link is the NXP single, unified debug probe architecture for all NXP general-purpose Arm Cortex-M based MCUs. MCU-Link is implemented on the latest evaluation boards (known as MCU-Link OB) of NXP. This firmware in this debug interface may be updated using the host computer utility called MCU-Link firmware. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

NXP provides the MCU-Link firmware, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link. The utility can be downloaded from MCU-Link Debug Probe Architecture.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux operating system, follow the similar steps.

1. Install the MCU-Link firmware.
2. Unplug the USB cable of the board.
3. Enable ISP mode on MCU-Link Core - LPC55xxx (install the jumper labeled ISP Enable).

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 firmware installation directory (<*firmware install dir*>).
    1. To program CMSIS-DAP debug firmware: <*firmware install dir*>*/scripts/program_CMSIS*
    2. To program J-Link debug firmware: <*LPCScrypt install dir*>*/scripts/program_JLINK*
6. Disable ISP mode on MCU-Link Core - LPC55xxx (remove the jumper installed in *Step 3*).
7. Repower the board by removing the USB cable and plugging it in again.

Parent topic:*Updating debugger firmware*

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.3 Getting Started with MCUXpresso SDK GitHub

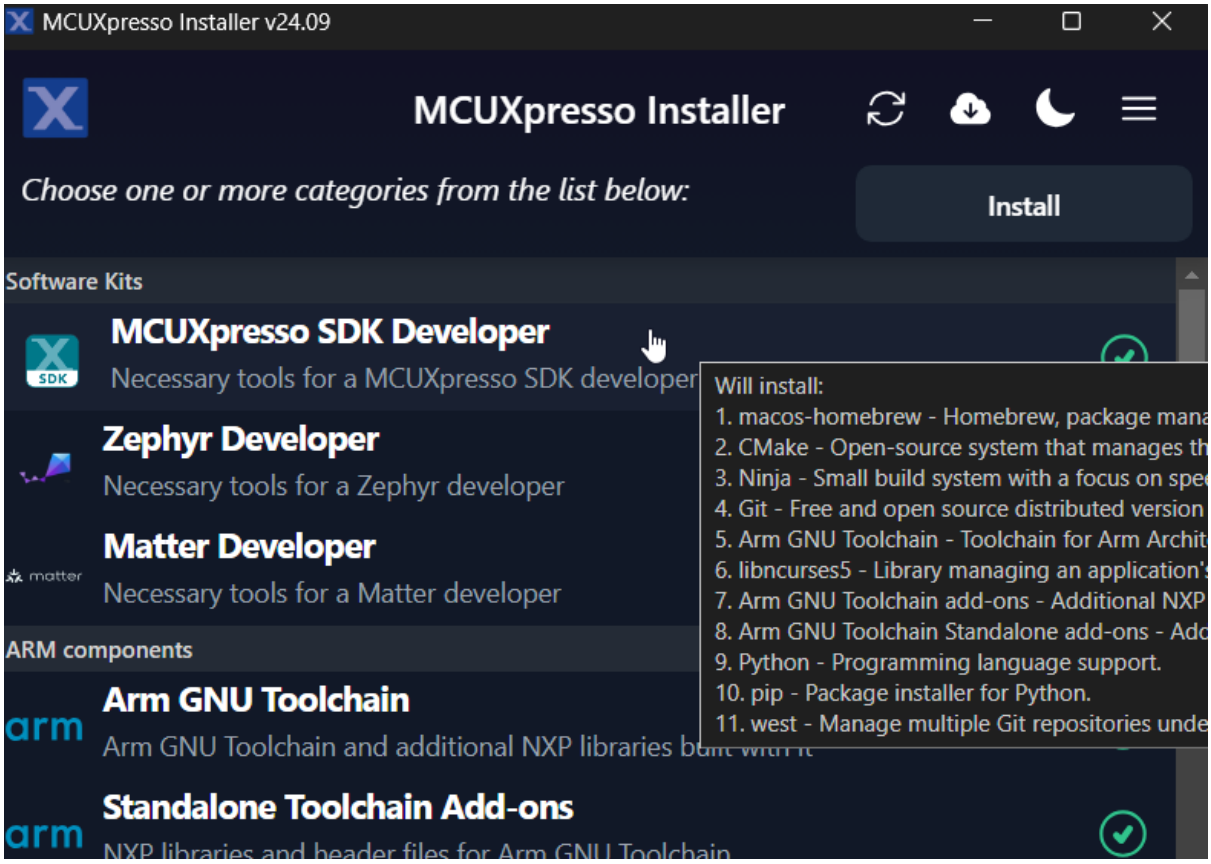
1.3.1 Getting Started with MCUXpresso SDK Repository

Installation

NOTE

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

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



Alternative: Manual Installation

Basic tools

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

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

Then configure your username and email using the commands:

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

Python Install python 3.10 or latest. Follow the Python Download guide.

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

West Please use the west version equal or greater than 1.2.0

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

Build And Configuration System

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

For Windows, you can directly use the .msi installer like cmake-3.31.4-windows-x86_64.msi to install.

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

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

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

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

For Linux, you can use your system package manager or you can directly download the ninja binary to work.

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

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

Please make sure *python* environment is setup ready then you can use the Kconfig.

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

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

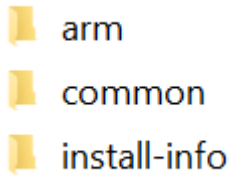
Toolchain MCUXpresso SDK supports all mainstream toolchains for embedded development. You can install your used or interested toolchains following the guides.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

Get MCUXpresso SDK Repo

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

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

# Update the west projects
cd mcuxpresso-sdk
west update

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

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

1. Navigate to the core directory:

   ```
   cd mcuxsdk
   ```

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

   ```
   python -m venv .venv

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

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

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

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

3. Install the required Python packages:

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

Explore Contents

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

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

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

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

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

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

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

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

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

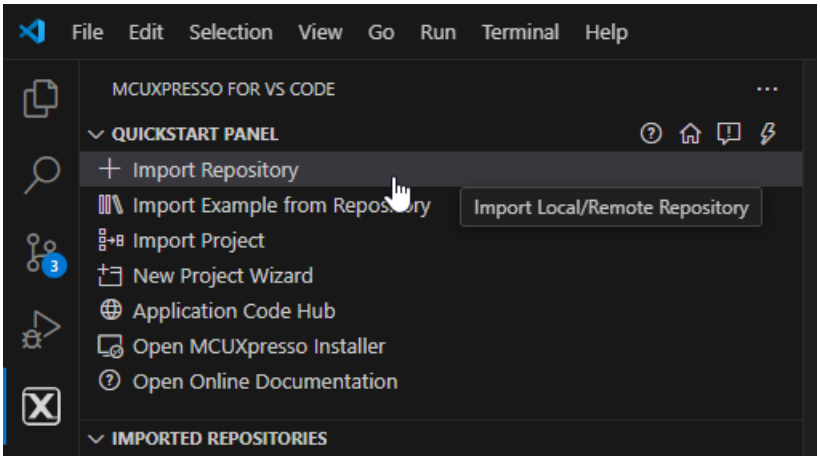
Run a demo using MCUXpresso for VS Code

This section explains how to configure MCUXpresso for VS Code to build, run, and debug example applications. This guide uses the hello_world demo application as an example. However, these steps can be applied to any example application in the MCUXpresso SDK.

Build an example application This section assumes that the user has already obtained the SDK as outlined in *Get MCUXpresso SDK Repo*.

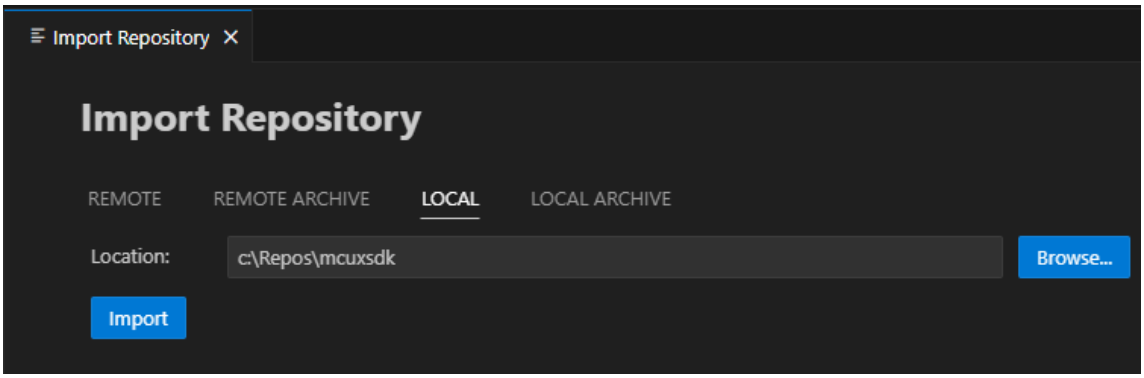
To build an example application:

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

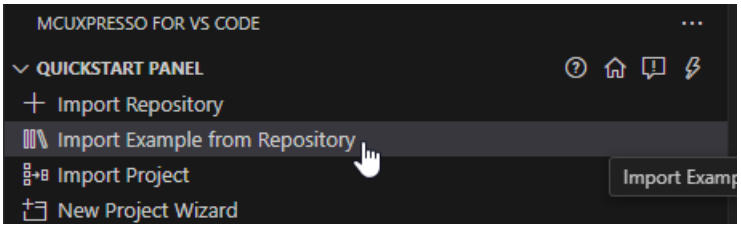


Note: You can import the SDK in several ways. Refer to MCUXpresso for VS Code Wiki for details.

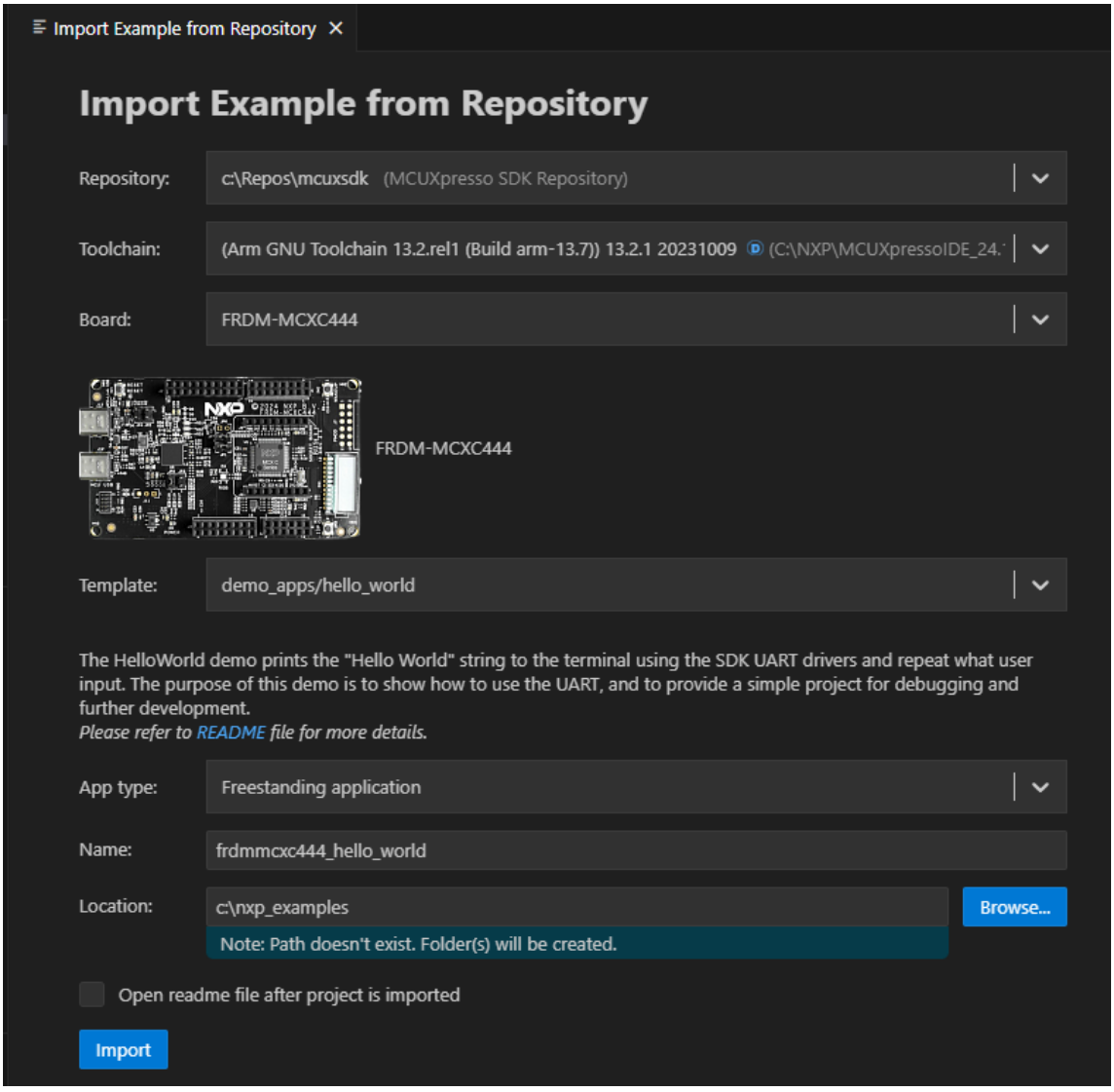
Select **Local** if you've already obtained the SDK as seen in *Get MCUXpresso SDK Repo*. Select your location and click **Import**.



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

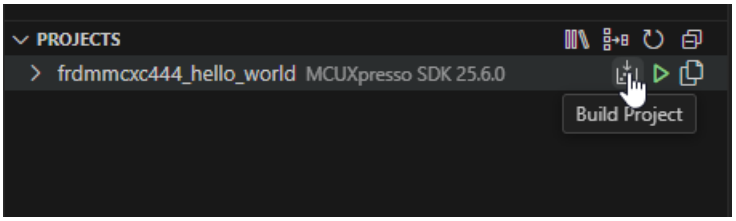


In the dropdown menu, select the MCUXpresso SDK, the Arm GNU Toolchain, your board, template, and application type. Click **Import**.

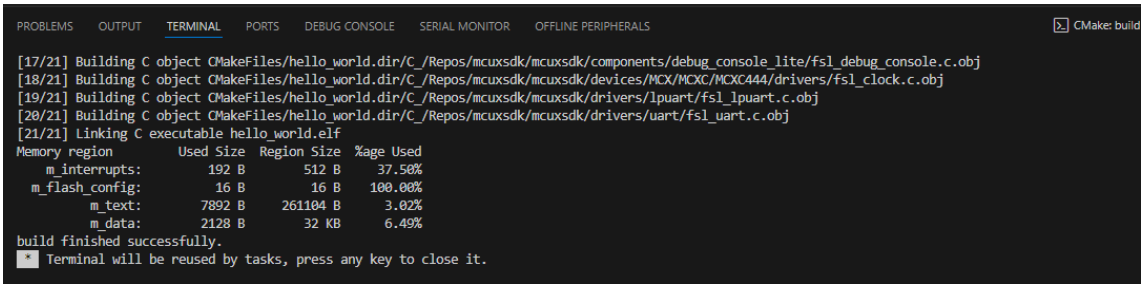


Note: The MCUXpresso SDK projects can be imported as **Repository applications** or **Free-standing applications**. The difference between the two is the import location. Projects imported as Repository examples will be located inside the MCUXpresso SDK, whereas Free-standing examples can be imported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

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

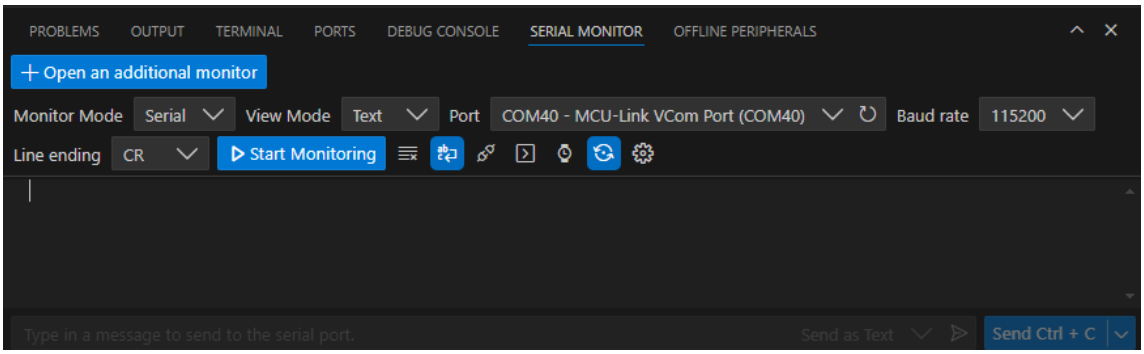


The integrated terminal will open at the bottom and will display the build output.

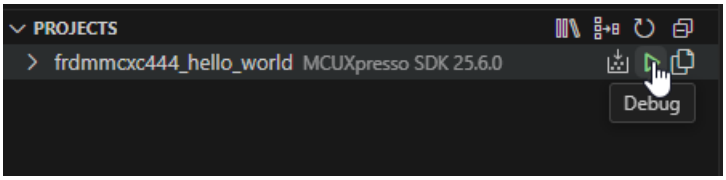


Run an example application **Note:** for full details on MCUXpresso for VS Code debug probe support, see MCUXpresso for VS Code Wiki.

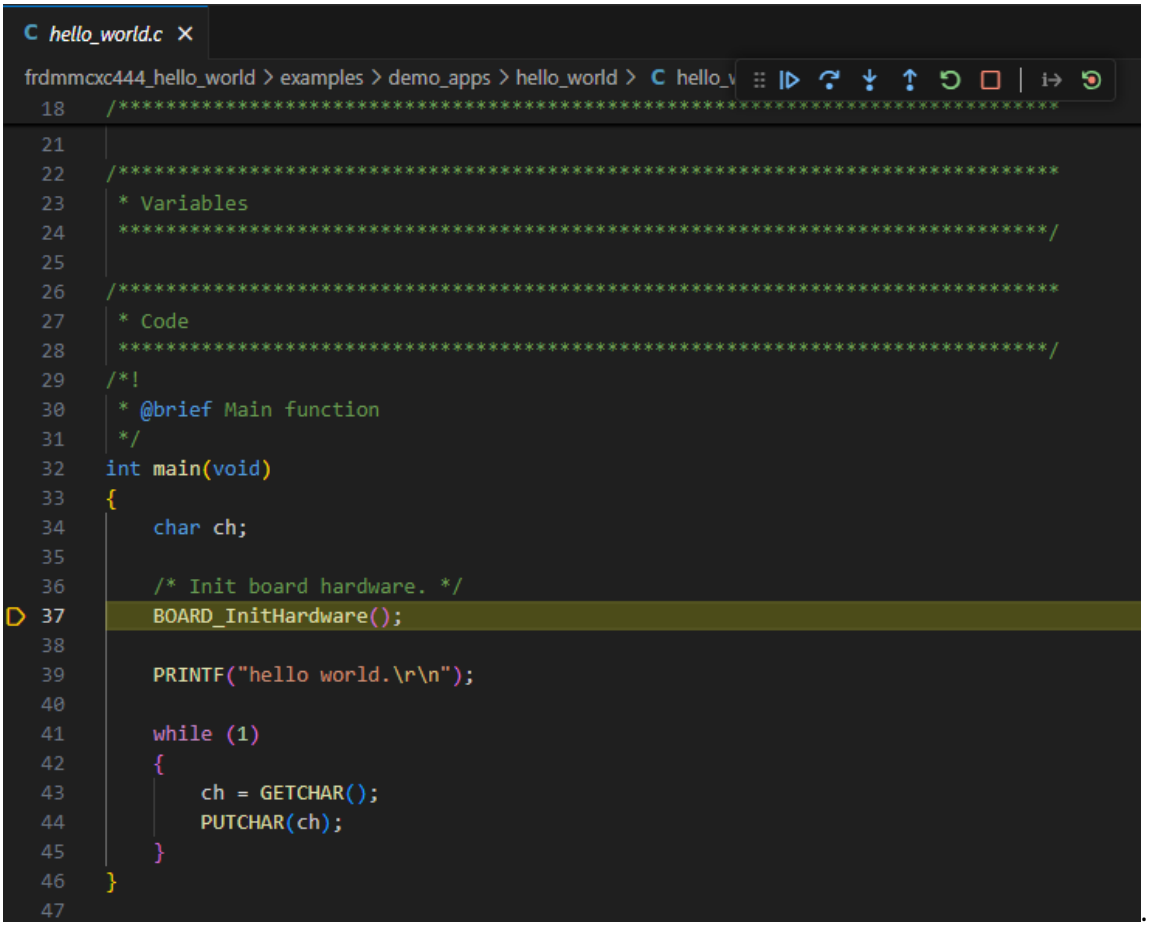
1. Open the **Serial Monitor** from the VS Code's integrated terminal. Select the VCom Port for your device and set the baud rate to 115200.



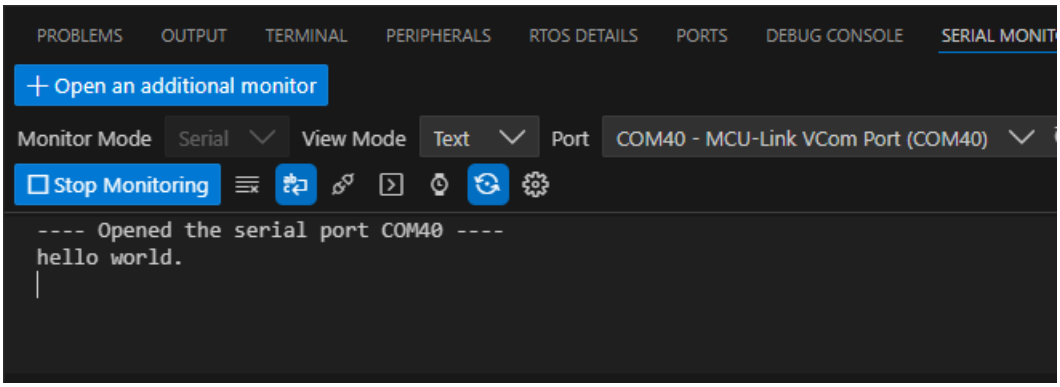
2. Navigate to the **PROJECTS** view and click the play button to initiate a debug session.



The debug session will begin. The debug controls are initially at the top.



3. Click **Continue** on the debug controls to resume execution of the code. Observe the output on the **Serial Monitor**.



Running a demo using ARMGCC CLI/IAR/MDK

Supported Boards Use the west extension `west list_project` to understand the board support scope for a specified example. All supported build command will be listed in output:

```
west list_project -p examples/demo_apps/hello_world [-t armgcc]

INFO: [  1][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evk9mimx8ulp -Dcore_id=cm33]
INFO: [  2][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkbimxrt1050]
INFO: [  3][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
```

(continues on next page)

(continued from previous page)

```
↪evkbmimxrt1060]
INFO: [  4][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkbmimxrt1170 -Dcore_id=cm4]
INFO: [  5][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkbmimxrt1170 -Dcore_id=cm7]
INFO: [  6][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkcmimxrt1060]
INFO: [  7][west build -p always examples/demo_apps/hello_world --toolchain armgcc --config release -b␣
↪evkmcimx7ulp]
...
```

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

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

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

Here are some typical usages for generating a SDK example:

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

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

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

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

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

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

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

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

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

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

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

For more details, please refer to System build.

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

1. Run cmake configuration

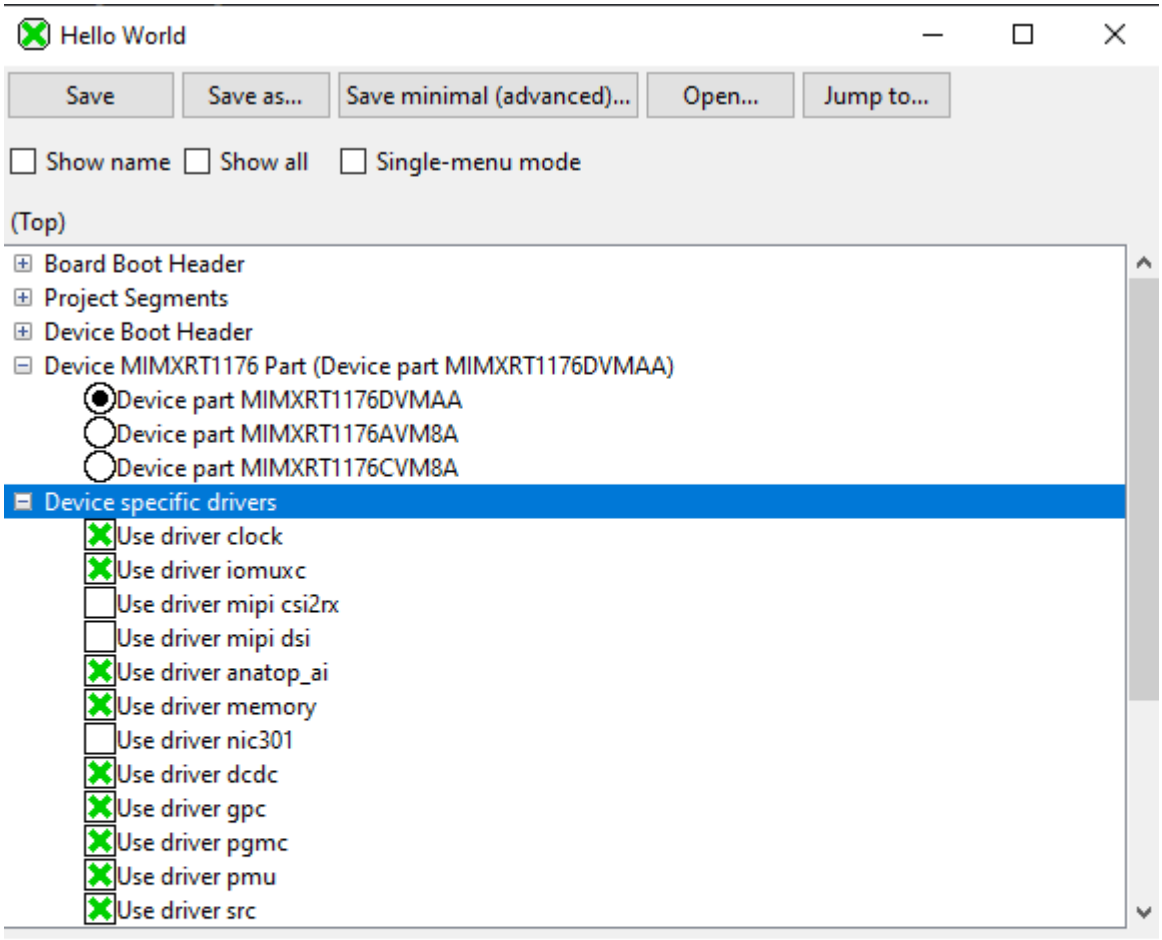
```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

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

2. Run guiconfig target

```
west build -t guiconfig
```

Then you will get the Kconfig GUI launched, like




```
Kconfig definition, with parent deps. propagated to 'depends on'
===============================================================

At D:/sdk_next/mcuxsdk\devices\../devices/RT/RT1170/MIMXRT1176\drivers/Kconfig:5
Included via D:/sdk_next/mcuxsdk/examples/demo_apps/hello_world/Kconfig:6 ->
D:/sdk_next/mcuxsdk/Kconfig.mcuxpresso:9 -> D:/sdk_next/mcuxsdk\devices/Kconfig:1
-> D:/sdk_next/mcuxsdk\devices\../devices/RT/RT1170/MIMXRT1176/Kconfig:8
Menu path: (Top)

menu "Device specific drivers"
```

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

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

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

Flash the hello_world example:

```
west flash -r linkserver
```

Debug Start a gdb interface by following command:

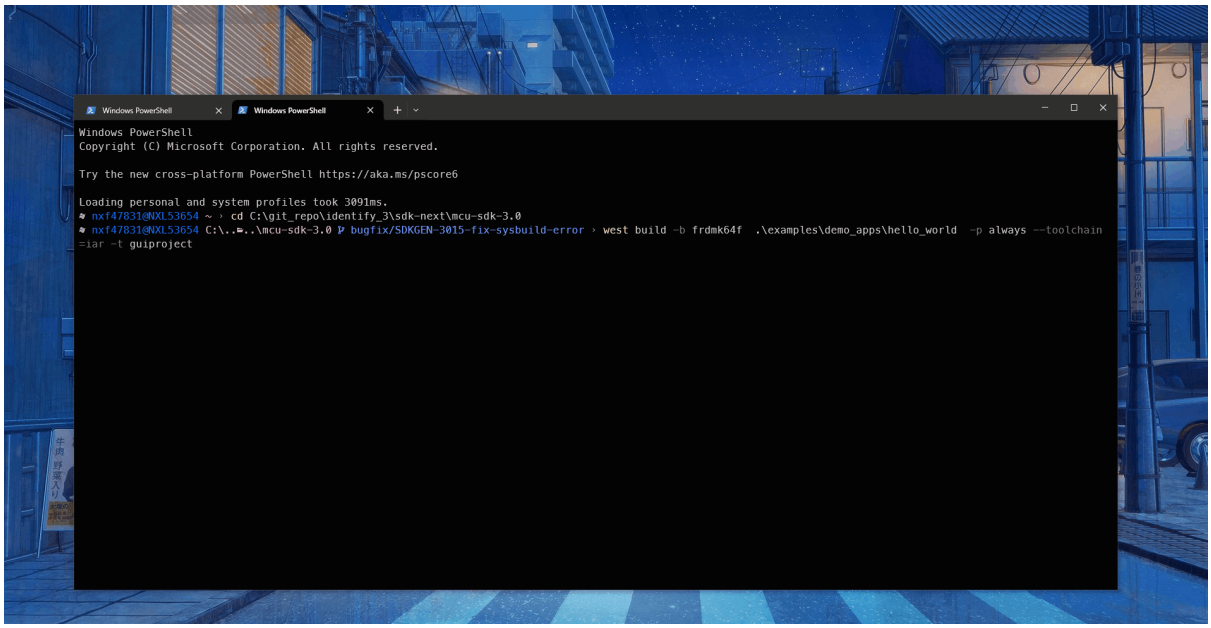
```
west debug -r linkserver
```

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

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

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

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



Note, please follow the *Installation* to setup the environment especially make sure that *ruby* has been installed.

1.4 Getting Started with MCUXpresso SDK Xplorer

1.4.1 Getting Started with MCUXpresso SDK Xplorer

Overview

Cadence Tensilica Xplorer is a complete development environment that helps users create application code for high-performance Tensilica processors. Xplorer is the interface to powerful

software development tools such as the XCC compiler, assembler, linker, debugger, code profiler, and full set of GUI tools.

Xplorer, including both GUI and command-line environment, is the only available development IDE for the DSP core of MIMXRT700.

Install Xplorer Toolchains

This section provides information on Xtensa Software Tools Platform Support and steps to:

- Install the Xtensa Xplorer IDE and Tools
- Install License Key
- Install RT700 DSP Build Configuration
- Install Xtensa On Chip Debugger Daemon
- Program MCU-Link Debug Probe with SEGGER J-Link (Optional)
- Install Xtensa Software Tools without IDE

Xtensa Software Tools Platform Support The Xtensa Software Tools are officially supported on the following platforms:

Windows: Win 10 64-bit, Win 8 64-bit, Win 7 64-bit.

Linux: RHEL 6 64-bit (with 'Desktop' package installed).

There may be compatibility issues with other versions of Linux or Windows, especially when using the IDE.

Note: The security-enhanced Linux (SELinux) is not a supported platform because the OS can prevent different shared libraries (including Xtensa Tools) from loading.

For more information on platform support and installation guidelines, see the Xtensa Development Tools Installation Guide.

Parent topic:*Install Xplorer Toolchains*

Install the Xtensa Xplorer IDE and Tools To install the Xtends Xplorer IDE and tools, perform the following steps:

1. Go to the URL https://tensilicatools.com/download/hifi-dsp-development-tools-for-rt700/ and log in. If you are accessing the site for the first time, make sure to register.
2. Make sure to use your corporate email address to register.



3. Once registered you should receive an email confirmation with an activation link from 'Tensilica Tools' no-reply@tensilicatools.com. Make sure to check the spam folder if this email does not show up in the inbox.
4. To complete the registration, click the activation link.
5. Once registered, log in to see the material available for download:



1. Download and install the **Xplorer IDE V10.1.11** for your operating system (Windows or Linux).
2. Download the **DSP Configuration** for your operating system – installed later through the IDE, see Install RT700 DSP Build Configuration. **Note:** NXP recommends version **10.1.11** of the Xtensa Xplorer IDE and tools for use with the RT700 DSP.

Parent topic:*Install Xplorer Toolchains*

Install License Key Xtensa development tools use FLEXlm for license management. FLEXlm licensing is required for tools such as the Xtensa Xplorer IDE, Xtensa C and C++ compiler.

Currently RT700 supports node-locked license for Xtensa tools. A node-locked license permits tools to run on a specific computer, tied to the MAC address of the primary network interface permanently attached to the machine.

Identify PC MAC Address To generate the correct license file, you should first identify the appropriate MAC for the computer you plan to run Xtensa tools on. Remove '-' or ':' symbols in the MAC address.

Windows:

```
C:\Users>ipconfig /all

Wireless LAN adapter Wireless Network Connection:

   Connection-specific DNS Suffix  . : us-sjo01.nxp.com
   Description . . . . . . . . . . . : Intel(R) Dual Band Wireless-AC 8265
   Physical Address. . . . . . . . . : 14-4F-8A-63-8C-33
   DHCP Enabled. . . . . . . . . . . : Yes
```

Linux:

```
[user@rhel ~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 12:34:56:78:90:AB
```

Note: For Linux: MAC address must be associated with eth0 interface. If not, FLEXlm cannot perform the license checkout, compilation, or simulation of code is not possible. If the host has the MAC address associated with another interface, for example em1, the following approach may be used.

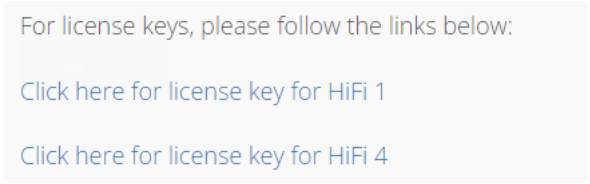
```
# Add udev rule for naming interface
$ sudo vim /etc/udev/rules.d/70-persistent-net.rules
# udev rule (replace 'XX' with the MAC address of your PC):
SUBSYSTEM==”net”, ACTION==”add”, ATTR{address}==”XX:XX:XX:XX:XX:XX”, NAME=”eth0”
# Change ”em1” to ”eth0” in your interfaces file.
$ sudo vim /etc/network/interfaces
# Restart udev or reboot machine
$ sudo reboot
```

Alternatively, you can use the approach recommended by your IT team to rename the interface to eth0.

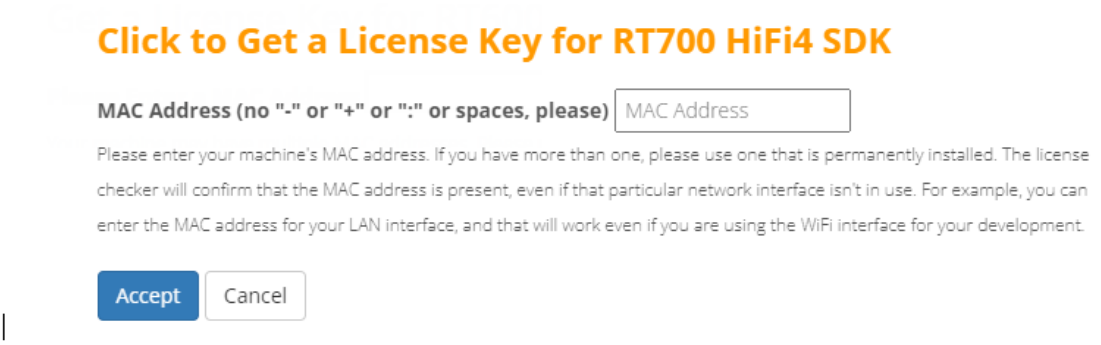
Parent topic: Install License Key

Download License Key To download the license key:

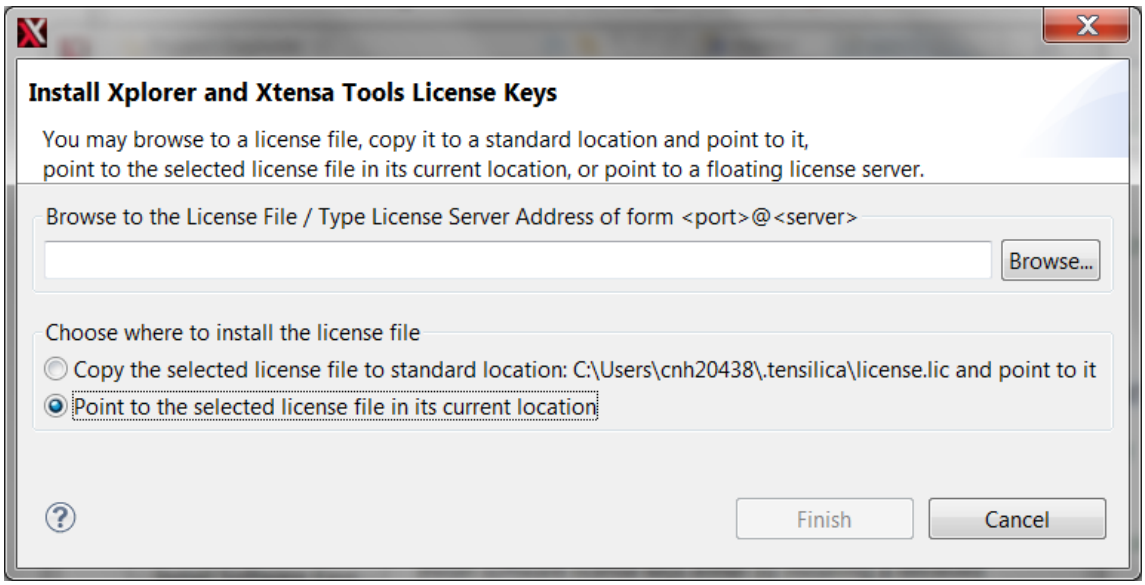
1. Reload or return to the Tensilica URL: https://tensilicatools.com/download/hifi-dsp-development-tools-for-rt700/.
2. Click the button and get the license key for RT700 SDK.

3. Provide the address.



4. Once the license file has been generated and downloaded, open your recently installed Xplorer V10.1.11.
5. Select menu **Help** > **Xplorer License Keys** > **Install Software Keys**.
6. Select the license key file.



7. Click the **Finish** button.

   Note: The generated license file only supports debug/run on the RT700 device target. It does not support software simulation/Xplorer ISS. If you have special requirements to run the software simulations, contact Cadence directly.

Parent topic: Install License Key

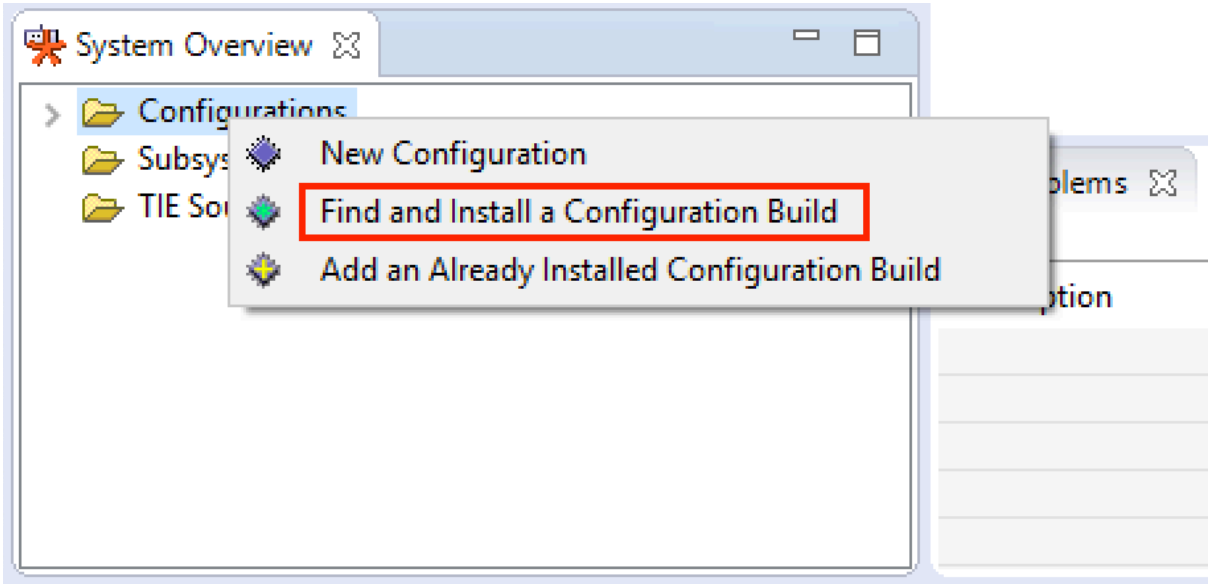
Parent topic:*Install Xplorer Toolchains*

Install RT700 DSP Build Configuration *'Build Configuration'* is a term that describes all parameters and necessary build includes for the Tensilica processor implementation for development. It is mandatory to install a specific build configuration before starting development on RT700.

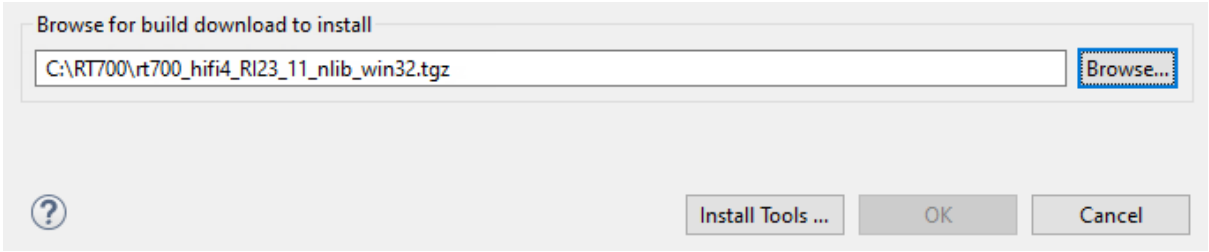
The build configuration is provided by NXP as a binary file and is imported into the Xplorer IDE. The binary file for the OS is available at the Tensilica URL: https://tensilicatools.com/download/rt700-download-page/.



The build configuration is installed into the IDE using the **System Overview** panel available in the lower left corner, by default. If the **System Overview** panel is not visible, toggle using the **Window** > **Show View** > **System Overview** menu item.



Click **OK** for build download to install.



|

|

Parent topic:*Install Xplorer Toolchains*

Install Xtensa On Chip Debugger Daemon The Xtensa On Chip Debugger Daemon (xt-ocd), is a gdb-based debugging tool but is not installed by default with the Xplorer IDE. For installation, a self-extracting executable installer is included with the IDE, which can be found at the following location:

Windows:

C:\usr\xtensa\XtDevTools\downloads\RI-2023.11\tools\xt-ocd-14.11-windows64-installer.exe

Linux:

~/xtensa/XtDevTools/downloads/RI-2023.11/tools/xt-ocd-14.11-linux64-installer

Currently, xt-ocd supports J-Link and Arm RVI/DSTREAM probes over Serial Wire Debug (SWD) for RT700. xt-ocd installs support for J-Link probes but does not install the required J-Link drivers which must be installed separately. Make sure that the latest version of J-Link software is installed.

Note: For Linux: When installing xt-ocd on Linux, ensure that the symlink is manually added to the installed J-Link driver: ln -s <jlink-install-dir>libjlinkarm.so.6 <xocd-install-dir>/modules/libjlinkarm.so.6.

xt-ocd is configured with an XML input file 'topology.xml' that modifies to fit the debugger hardware. Using J-link as an example, use the content below to replace the original template for HiFi4 DSP

Note: It is mandatory to replace the 'usbser' section with the J-Link serial number on the back of the J-Link hardware in use.

```
<configuration>
 <controller id='Controller0' module='jlink' usbser='600116164' type='swd' speed='1000000' locking='1'/>
 <driver id='XtensaDriver0' dap='1' xdm-offset='0x80020000' module='xtensa' step-intr='mask,stepover,setps
↪' />
 <chain controller='Controller0'>
 <tap id='TAP0' irwidth='4' />
 </chain>
 <system module='jtag'>
 <component id='Component0' tap='TAP0' config='trax' />
 </system>
 <device id='Xtensa0' component='Component0' driver='XtensaDriver0' ap-sel='3'/>
 <application id='GDBStub' module='gdbstub' port='20000' sys-reset='0'>
 <target device='Xtensa0' />
 </application>
</configuration>
```

Below is another topology.xml example for HiFi1 DSP.

```
<configuration>
      <controller id='Controller0' module='jlink' usbser='600116164' type='swd' speed='1000000' locking='1
```

(continues on next page)

(continued from previous page)

```
↪'/>

 <driver id='XtensaDriver0' dap='1' xdm-offset='0x80030000' module='xtensa' step-intr='mask,stepover,
↪setps' />
 <chain controller='Controller0'>
  <tap id='TAP0' irwidth='4' />
 </chain>
 <system module='jtag'>
  <component id='Component0' tap='TAP0' config='trax' />
 </system>
 <device id='Xtensa0' component='Component0' driver='XtensaDriver0' ap-sel='3'/>
 <application id='GDBStub' module='gdbstub' port='20000' sys-reset='0'>
  <target device='Xtensa0' />
 </application>
</configuration>
```

Congratulations! All Xplorer toolchains are installed.

For more details on Xtensa software tools, build configurations, or xt-ocd daemon, see the full set of documents in Xplorer menu **Help** > **PDF Documentation**.

Parent topic:*Install Xplorer Toolchains*

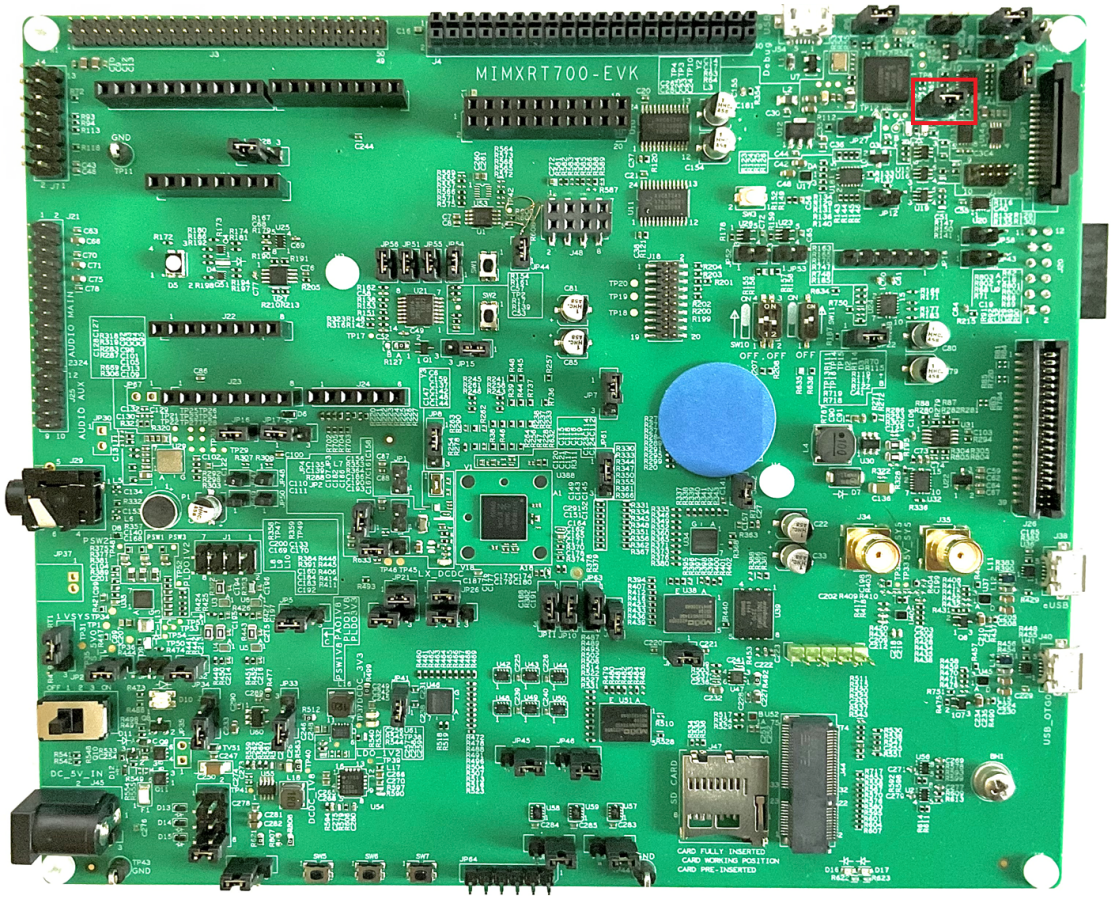
Program MCU-Link Debug Probe with SEGGER J-Link (Optional) In addition to standalone probes, the onboard MCU-Link debug probe can be used to debug HiFi4 DSP over SWD port. The RT700 EVK has an LPC55S69 MCU (top right corner on EVK) and by default has been pre-programmed as CMSIS-DAP probe. CMSIS-DAP is not compatible with HiFi4 the DSPs, but to enable serial logging of Core 1, CMSIS-DAP firmware is required.

To use the on-board J-Link without Core 1 serial logging, a few extra steps are required to flash it with J-Link firmware.

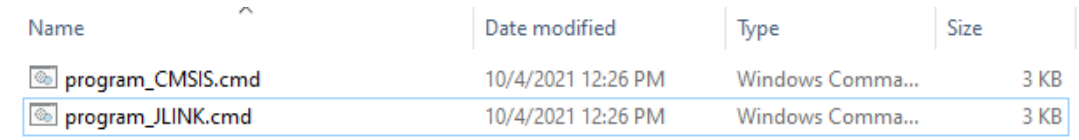
The steps are:

1. Install MCU-Link, a command-line tool for programming onboard MCU-Link debug probe with the CMSIS-DAP and J-Link firmware. MCU-Link installer is available for download from: https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcu-link-debug-probe:MCU-LINK#t996.
2. After the download is complete, run the installer.

   Note: During the installation, the DFU, and VCOM drivers are automatically installed for all platforms.
3. To update the MCU-Link debug circuit firmware, unplug the USB cable on J54 and then connect the JP20 jumper. .



4. Reconnect the board to the host computer over the debug link USB connector J54.
5. Go to the scripts folder in the MCU-Link install location: C:\NXP\MCU-LINK_installer_2.263\scripts.

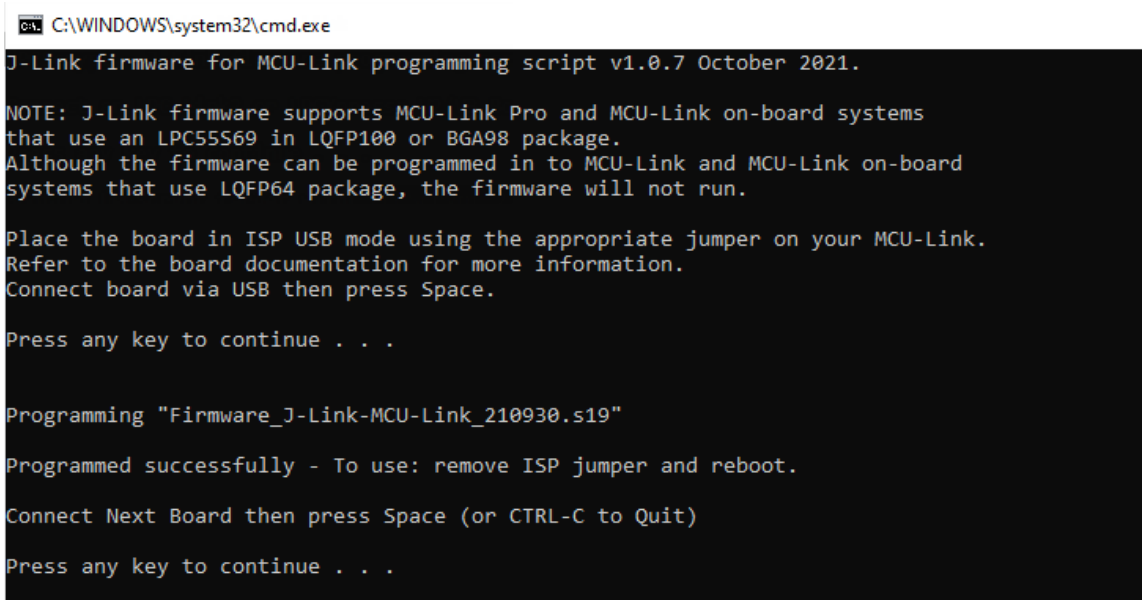


6. In that command shell, run the program_JLINK script to install the JLink debug firmware.

```
<Intall Dir>scripts\program_JLINK
```

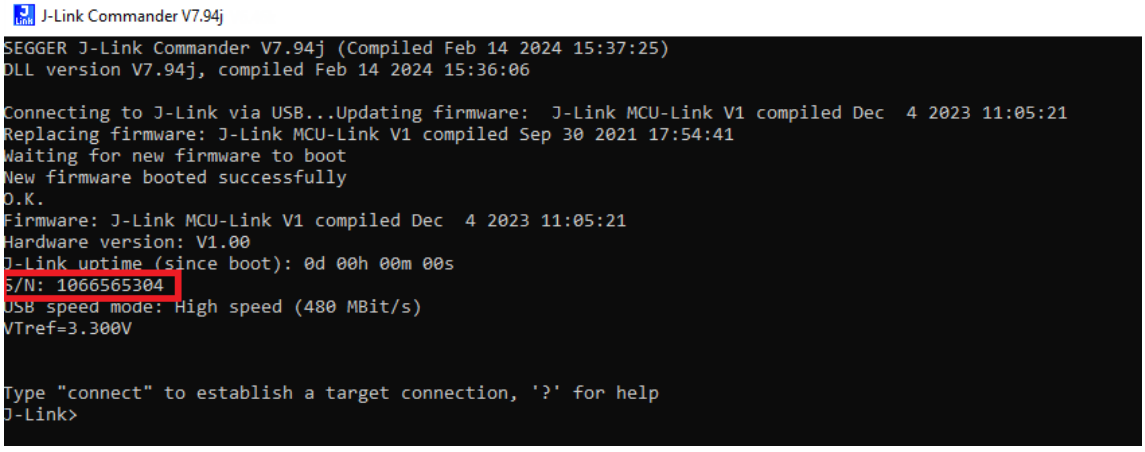
Note: File paths in this document use the Windows directory separator, on Linux or Mac OSX. The file paths must be replaced with '/.' For Windows users, shortcuts to the scripts are available from the MCU-LInk entry on the Start menu.

7. Verify once you select the firmware (in this case J-Link), MCU-Link. The console appears as shown in Figure 4.

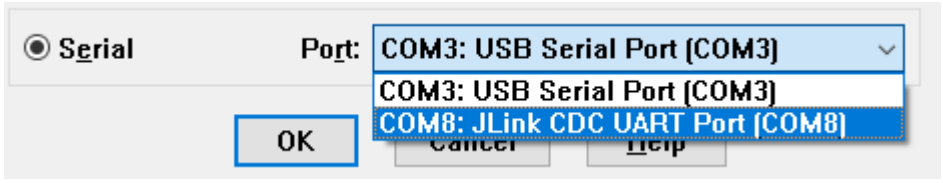


8. Open/ Disconnect JP20 and power cycle the board. The onboard MCU-Link is ready to be used as SEGGER J-Link probe.

   Every EVK/ MCU-Link has a different J-Link S/N. Therefore, make sure to write down the S/N for xt-ocd and topology.xml as indicated in Install Xtensa On Chip Debugger Daemon.



9. MCU-Link debug probe creates a virtual serial port over USB, so the extra UART2USB cable for debugging is not required.



The download link provides the document, demo videos and details on MCU-Link. If you have any questions, or have difficulties to program the probe, see https://www.nxp.com/document/guide/getting-started-with-the-mcu-link:GS-MCU-LINK.

Parent topic:*Install Xplorer Toolchains*

Install Xtensa Software Tools without IDE The Xtensa Software Tools can also be installed without the use of the IDE. The installation without the IDE is useful in a command-line only Linux environment, or for better compatibility with an unsupported Linux environment.

The command-line tools package is available as a redistributable zip file that is extracted with an Xplorer IDE install. To gain access to the tools package, the IDE must be installed once in the organization. The tools package is available at: ~/xtensa/XtDevTools/downloads/RI-2023.11/tools/XtensaTools_RI_2023_11_linux.tgz..

With the tools package and the DSP Build Configuration package available from the Tensilica Tools download site, the toolchain can be set up as follows:

```
# Create Xtensa install root
mkdir -p ~/xtensa/tools
mkdir -p ~/xtensa/builds
# Set up the configuration-independent Xtensa Tool:
tar zxvf XtensaTools_RI_2023_11_linux.tgz -C ~/xtensa/tools
# Set up the configuration-specific core files:
tar zxvf nxp_rt700_RI23_11_newlib_linux_redist.tgz -C ~/xtensa/builds
# Install the Xtensa development toolchain:
cd ~/xtensa
./builds/RI-2023.11-linux/nxp_rt700_RI23_11_newlib/install \
--xtensa-tools./tools/RI-2023.11-linux/XtensaTools \
--registry ./tools/RI-2023.11-linux/XtensaTools/configg
```

Parent topic:*Install Xplorer Toolchains*

Install MCUXpresso SDK

This section list the steps to: Download MCUXpresso SDK for RT700.

Download MCUXpresso SDK for RT700 DSP enablement for RT700, including drivers, middleware libraries, and demo applications are included with the latest RT700 SDK will be available for download from https://mcuxpresso.nxp.com. If you are accessing the site for the first time, make sure to register.

Parent topic:*Install MCUXpresso SDK*

Run and Debug DSP Demo using Xplorer IDE

This section lists the steps to:

- Prepare Arm Core for 'Hello World'
- Start Xtensa Debugger Daemon
- Prepare DSP Core for 'Hello World'

Prepare Arm Core for 'Hello World' Each of the DSP demos included in the MCUXpresso SDK consists of two separate applications that run on the Arm core and DSP core. The Arm core application initializes the DSP core and executes other application-specific functionality.

To debug the 'Hello World' DSP application, you must first set up and execute the Arm application using an environment of your choice. It is required to modify the global preprocessor macro, DSP_IMAGE_COPY_TO_RAM, to control loading of the DSP binary application.

When this macro is set to '0'/FALSE, the Arm application and DSP application are independently loaded and debugged.

- Build and execute the 'Hello World' Arm demo located in: <SDK_ROOT>/boards/mimxrt700evk/dsp_examples/hello_world_usart/cm33/.

Preparing an Arm core development environment is outside the scope of this document.

Note: If you are using MCUXpresso, it is highly recommended to upgrade to latest version of the SDK and match the latest EVK board.

Parent topic:*Run and Debug DSP Demo using Xplorer IDE*

Start Xtensa Debugger Daemon To debug DSP applications on RT700, ensure that the xt-ocd daemon is running. This application runs a gdb server that the Xtensa core debugger connects to.

Go to the command-line window and change directory (cd) to xt-ocd daemon installation path. The default path on Windows is C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.11.

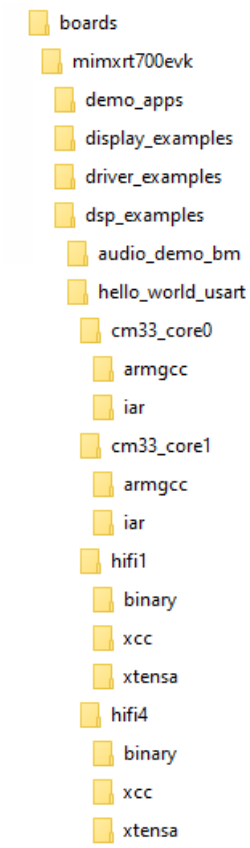
Execute the daemon with the custom topology.

```
**xt-ocd.exe -c topology.xml**
XOCD 14.11 2023-04-06 14:27:20
(c) 1999-2024 Cadence Design Systems Inc. All rights reserved.
[Debug Log 2024-02-19 22:41:53]
Loading module "gdbstub" v2.0.0.12
Loading module "jlink" v2.0.2.0
Using JLINK lib v.79410
Jlink USB Serial Number: 851006054
Connected to Jlink Device:
  Name:'SEGGER J-Link'
  S/N:851006054
  Firmware: J-Link V11 compiled Dec  4 2023 10:22:45
  Requested/Set TCK: 1000kHz/1000kHz
Jlink: Select pipelined SWD
SWD-DP with ID 0x6BA02477
Loading module "jtag" v2.0.0.20
Loading module "xtensa" v2.0.0.48
Starting thread 'GDBStub'
Opened GDB socket at port 20000
Initialize XDM driver
Warning: Warning: DAP Reset request failed! Ignoring...
```

Note: Some warning messages are expected and can be ignored. If you receive an error initializing the XDM driver, initialize and start the DSP core before debugging. For details on initializing and debugging, see Initialize DSP Core and Link DSP Profiles.

Parent topic:*Run and Debug DSP Demo using Xplorer IDE*

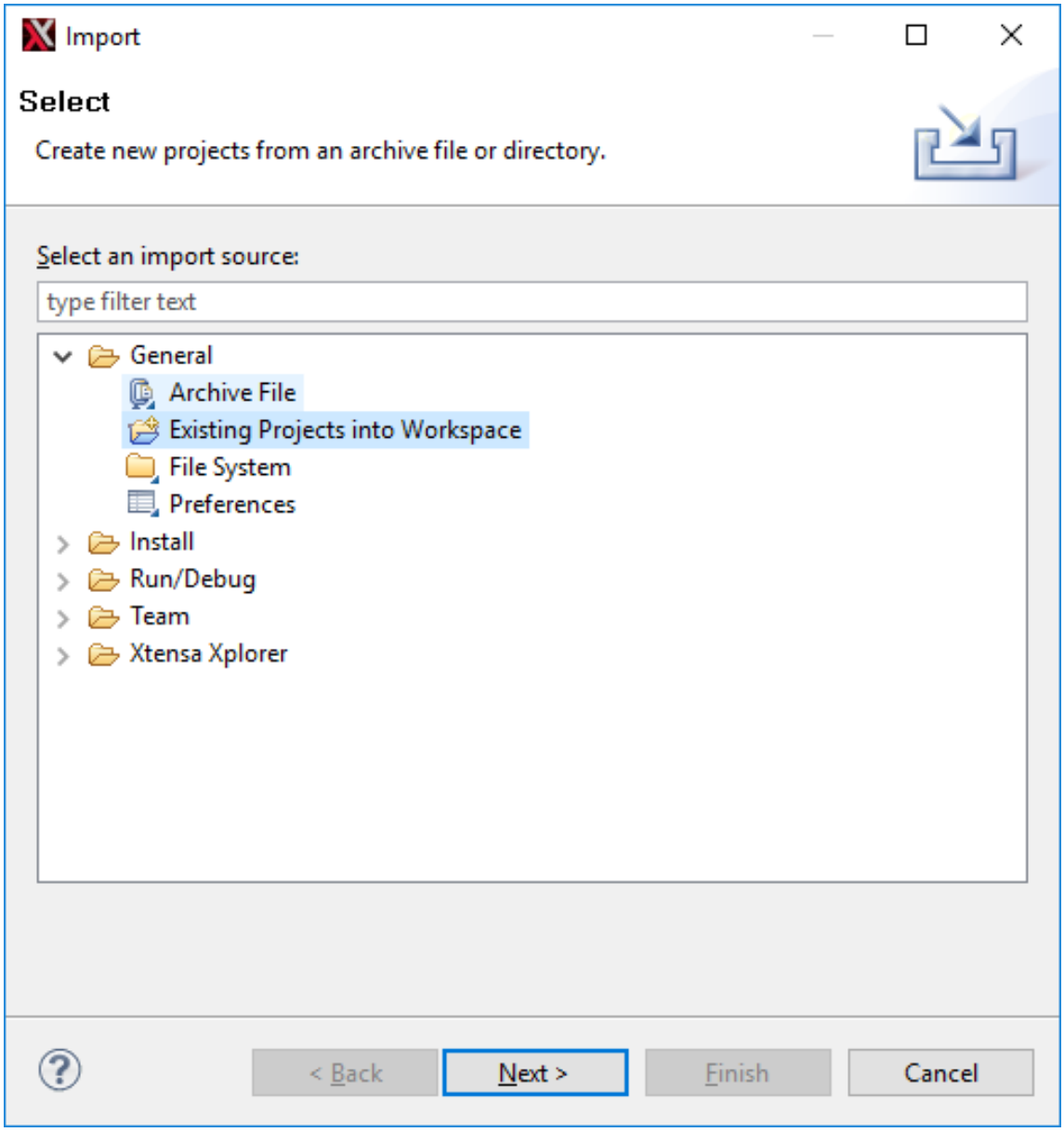
Prepare DSP Core for 'Hello World' The RT700 SDK provides a collection of DSP example applications located in boards/mimxrt700evk/dsp_examples/. Each DSP example has two four source directories for cm33_core0, cm33_core1, hifi1, and hifi4 cores. In this guide, HiFi4 example will be explained, but HiFi1 examples can be run with the same steps.



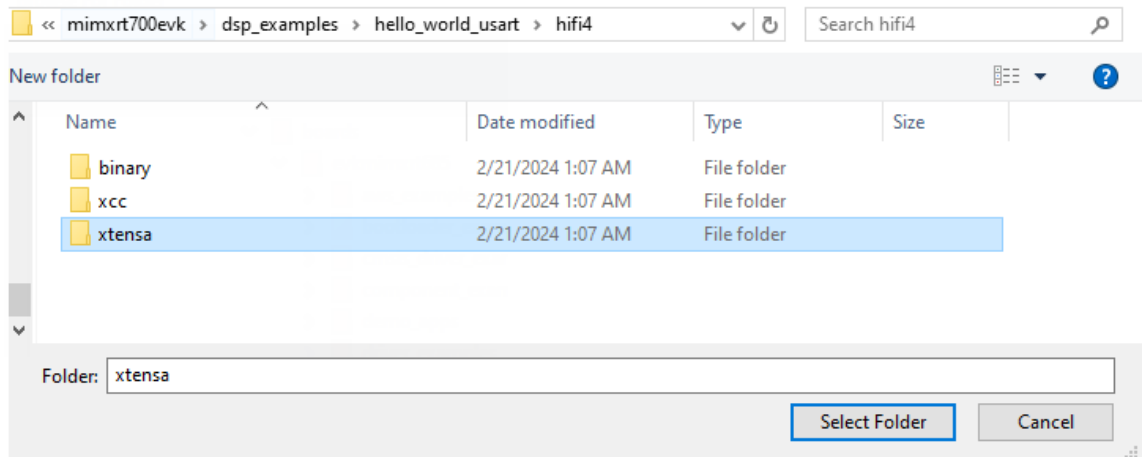
The projects for different supported toolchains are built in the above directories. For the DSP example above, the 'xcc' project builds on the command line and the 'xtensa' directory is an Xplorer IDE project.

To run the 'Hello World' demo, import the SDK sources into the Xplorer IDE.

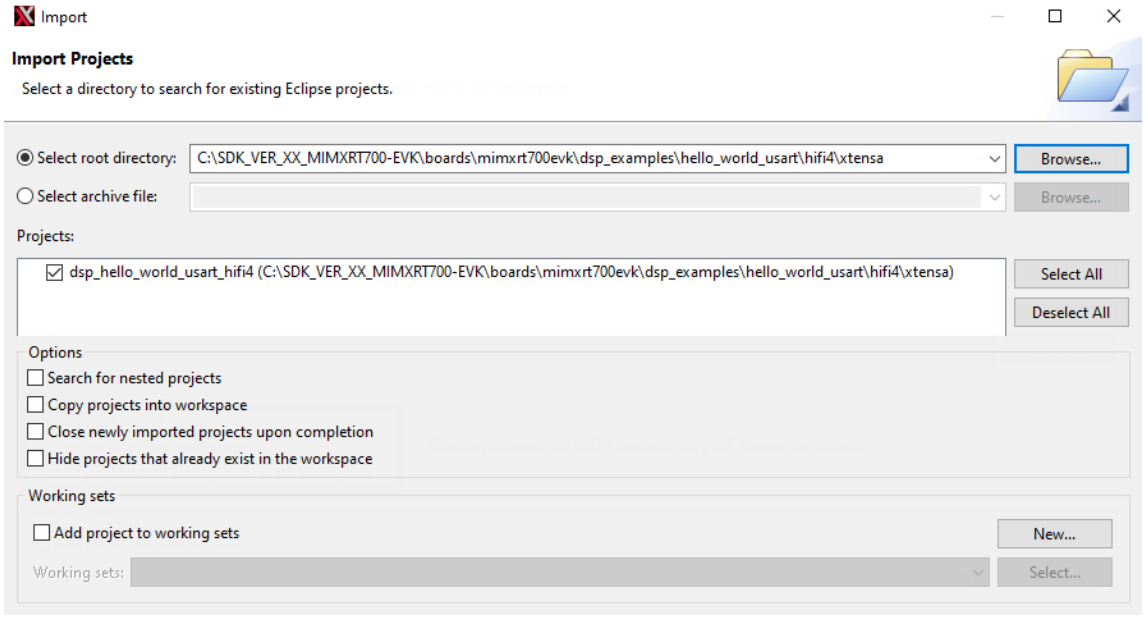
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Select the SDK directory <SDK_ROOT>\boards\mimxrt700evk\dsp_examples\hello_world_usart\hifi4\xtensa as root directory.

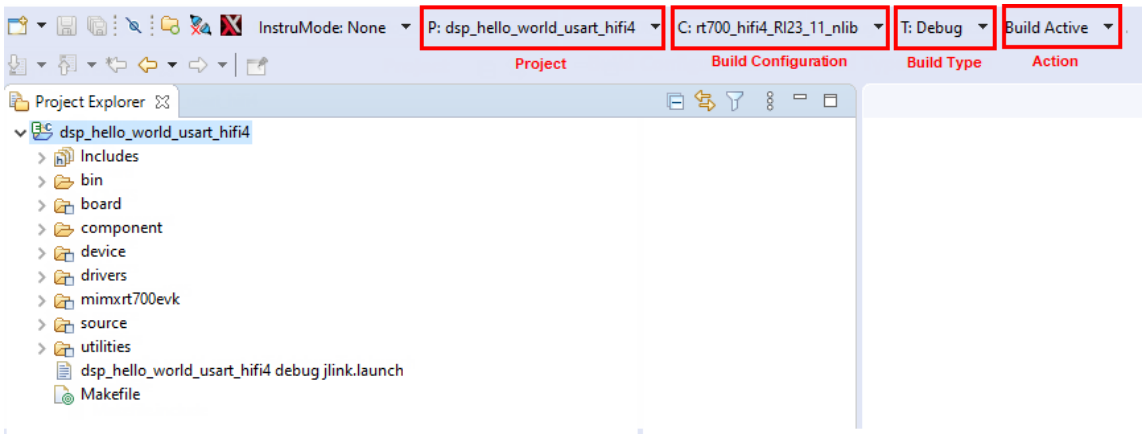


3. Leave all the other check boxes blank.

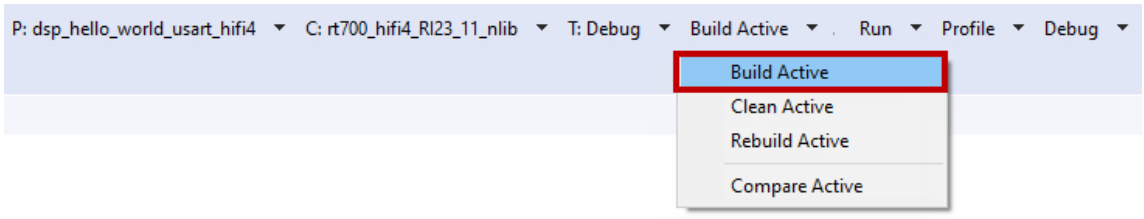


Once imported, the 'dsp_hello_world_usart_hifi4' appears in the Project Explorer.

4. To make a build selection for the project and hardware target configuration, use the drop-down buttons on the menu bar.

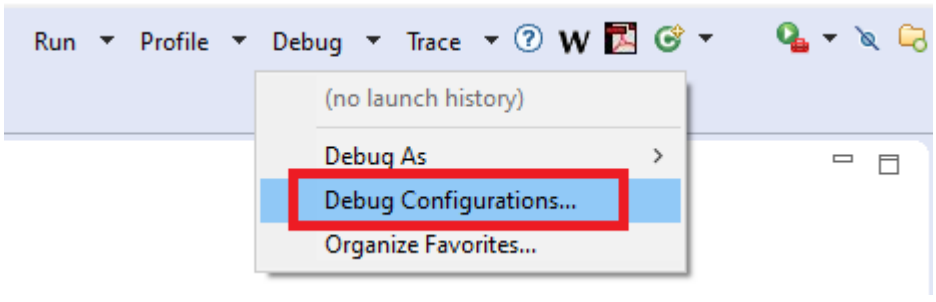


5. To build the project for debug, profile, or trace, use the action buttons on the right side of the menu bar.



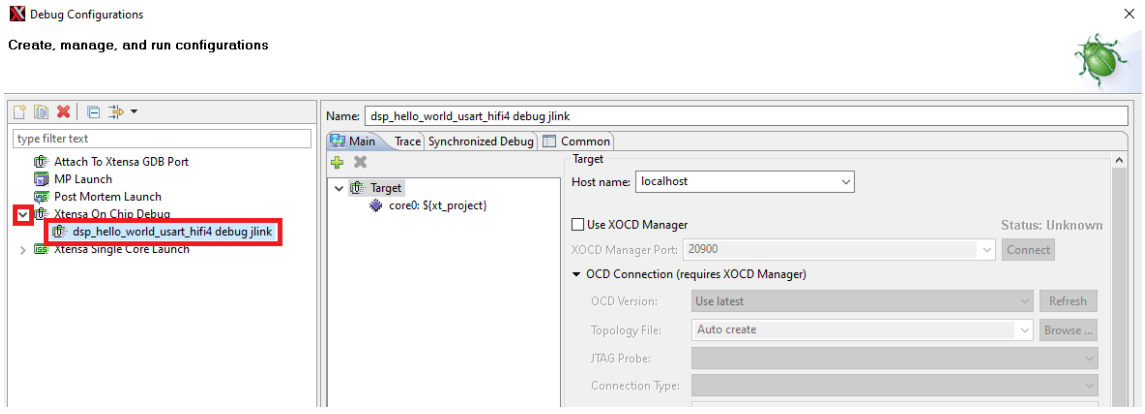
A default debug configuration is provided by the SDK project, which utilizes the on-chip debugger.

6. To choose the configuration, select the **Debug Configurations** menu item.

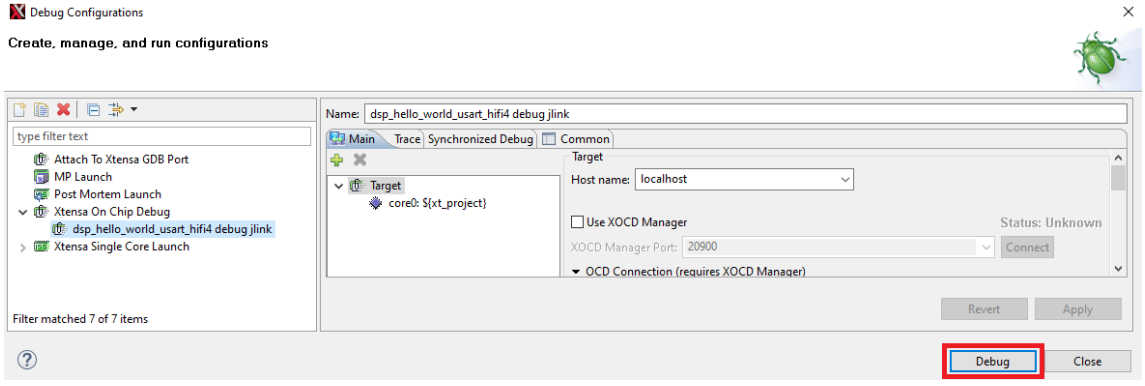


The **Debug Configurations** dialog box appears.

7. Expand the **Xtensa On Chip Debug** tree option and select '*dsp_hello_world_hifi4_debug_jlink*'.

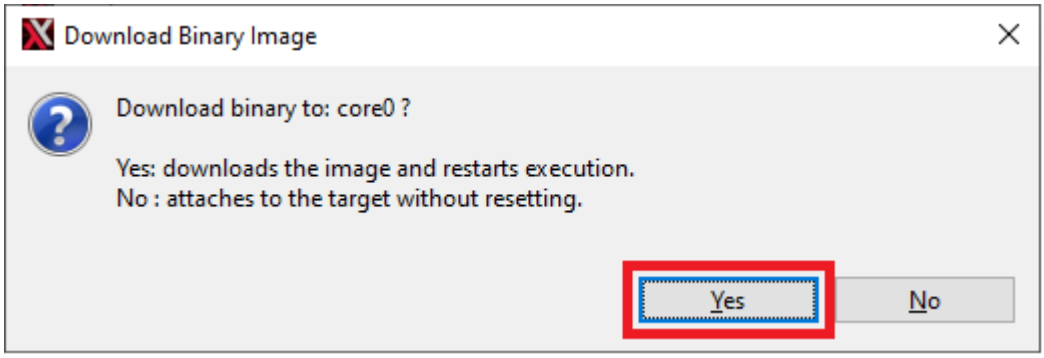


8. Click the **Debug** button. The actual debug on the chip initiates.



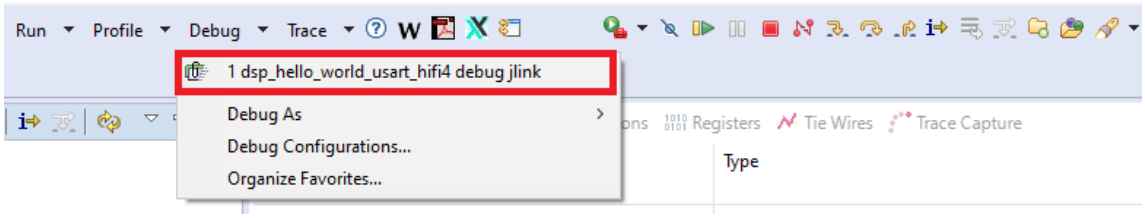
The Xplorer IDE prompts whether binaries should download to the hardware.

9. Select **Yes**.



10. The Xplorer IDE transitions to the **Debug** perspective after the binary download.

    Note: After initial configuration, it is possible to select the same debug configuration as in Figure 8.



The **Debug** perspective appears as in [Figure 9](prepare_dsp_core_for_hello_world.md#GVDVWQC).

|

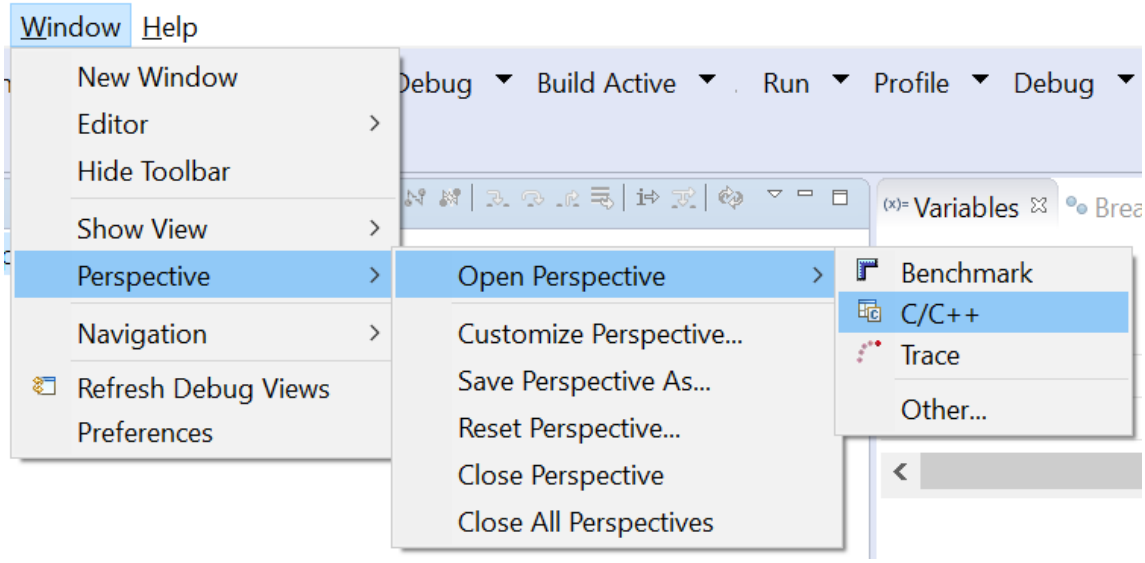
The program stops at the start of main\(\) function. To run the program, click the **Resume** or **Stepping through** icon button as shown in [Figure 10](prepare_dsp_core_for_hello_world.md#LKWEUGG).

|

11. After resuming / stepping through the 'printf' statement, the following output appears in the **Console** view of the IDE.

```
Hello World running on DSP core 'rt700_hifi4_RI23_11_nlib'
```

    To return to the default IDE layout after the debugging completes, select the previous code perspective.



Parent topic:*Run and Debug DSP Demo using Xplorer IDE*

1.5 Release Notes

1.5.1 MCUXpresso SDK Release Notes

Overview

The MCUXpresso SDK is a comprehensive software enablement package designed to simplify and accelerate application development with Arm Cortex-M-based devices from NXP, including its general purpose, crossover and Bluetooth-enabled MCUs. MCUXpresso SW and Tools for DSC further extends the SDK support to current 32-bit Digital Signal Controllers. The MCUXpresso SDK includes production-grade software with integrated RTOS (optional), integrated enabling software technologies (stacks and middleware), reference software, and more.

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

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

MCUXpresso SDK

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

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

Development tools

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

- MCUXpresso IDE, Rev. 25.06.xx
- IAR Embedded Workbench for Arm, version is 9.60.4
- Keil MDK, version is 5.42
- MCUXpresso for VS Code v25.09
- GCC Arm Embedded Toolchain 14.2.x
- Xtensa Xplorer, version is 10.1.11
- Xtensa C Compiler, version is RI-2023.11
- RISCV LLVM 19.1.7

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
MIMXRT700-EVK	MIMXRT735SGAWBR, MIMXRT735SGFOB, MIMXRT758SGAWBR, MIMXRT758SGFOB, MIMXRT798SGAWBR, **MIMXRT798SGFOB**

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

Wireless EdgeFast Bluetooth PAL For more information, see the MCUXpresso SDK EdgeFast Bluetooth Protocol Abstraction Layer User's Guide.

coreHTTP coreHTTP

openvg OpenVG library for devices with graphics acceleration hardware

NXP Wi-Fi The MCUXpresso SDK provides driver for NXP Wi-Fi external modules. The Wi-Fi driver is integrated with LWIP TCPIP stack and demonstrated with several network applications (iperf and AWS IoT).

For more information, see Getting Started with NXP based Wireless Modules and i.MX RT Platform Running on RTOS (document: UM11441).

Voice intelligent technology library Voice Intelligent Technology (VIT) Library provides wake word and voice command engine for voice control

VG-Lite GPU Library VGLite library for devices with VGLite graphics hardware acceleration engine

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

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

TinyCBOR Concise Binary Object Representation (CBOR) Library

PSA Test Suite Arm Platform Security Architecture Test Suite

SDMMC stack The SDMMC software is integrated with MCUXpresso SDK to support SD/MMC/SDIO standard specification. This also includes a host adapter layer for bare-metal/RTOS applications.

PKCS#11 The PKCS#11 standard specifies an application programming interface (API), called "Cryptoki," for devices that hold cryptographic information and perform cryptographic functions. Cryptoki follows a simple object based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a "cryptographic token".

Multicore Multicore Software Development Kit

MCU Boot Open source MCU Bootloader.

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.

eIQ The package contains several example applications using the eIQ TensorFlow Lite for Microcontrollers library.

eIQ machine learning SDK containing:

- Arm CMSIS-NN library (neural network kernels optimized for Cortex-M cores)
- Inference engines:
  - TensorFlow Lite Micro
  - DeepView RT
- Example code for TensorFlow Lite Micro, Glow, and DeepView RT

LVGL LVGL Open Source Graphics Library

llhttp HTTP parser llhttp

LittleFS LittleFS filesystem stack

JPEG library JPEG library

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

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

emWin The MCUXpresso SDK is pre-integrated with the SEGGER emWin GUI middleware. The AppWizard provides developers and designers with a flexible tool to create stunning user interface applications, without writing any code.

DSP Neural Networks DSP Neural Networks Framework based on Xtensa Neural Networks Library from Cadence Design Systems for Xtensa HiFi Audio Engines.

NatureDSP for Hifi4 Digital Signal Processing for Xtensa Hifi4 DSP Audio Engines.

NatureDSP for Hifi1 Digital Signal Processing for Xtensa Hifi1 DSP Audio Engines.

DSP Audio Streamer DSP Audio Streamer Framework based on Xtensa Audio Framework from Cadence Design Systems for Xtensa DSP Audio Engines.

DSP Codecs Codecs for Cadence DSPs

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

NXP ELS PKC ELS PKC crypto library

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

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.

XAF_playback limitation

SBC decoder is not working on the hifi4 core of RT700.

1.6 ChangeLog

1.6.1 MCUXpresso SDK Changelog

Board Support Files

board

[25.09.00]

- Improvement
  - Added BOARD_SetDeepSleepPinConfig and BOARD_RestoreDeepSleepPinConfig APIs.

[25.06.00]

- Initial version

clock_config

[25.06.00]

- Initial version

pin_mux

[25.06.00]

- Initial version

ACMP

[2.4.0]

- New Feature
  - Supported the plateforms which don't have continuous mode.

[2.3.0]

- Improvements
  - Expose C0 register FILTER_CNT bitfield and FPR bitfield to the user.

[2.2.0]

- Improvements
  - Updated feature macros for roundrobin mode, window mode, filter mode, and 3V domain removes.

[2.1.0]

- New Feature
  - Supported the plateforms which don't have hysteresis mode.

[2.0.6]

- Bug Fixes
  - Fixed the wrong comments, the DAC value should range from 0 to 255.

[2.0.5]

- Bug Fixes
  - Fixed the out-of-bounds error of Coverity caused by missing an assert sentence to avoid the return value of ACMP_GetInstance() exceeding the array bounds.
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.1, 14.4, 16.4, 17.7.

[2.0.4]

- Bug Fixes
  - Avoided changing w1c bit in ACMP_SetRoundRobinPreState().

[2.0.3]

- New Features
  - Added feature functions for usage of different power domains(1.8 V and 3 V). These functions are first enabled in ULP1. They are about:
    - ACMP_EnableLinkToDAC()
    - ACMP_SetDiscreteModeConfig()
    - ACMP_GetDefaultDiscreteModeConfig()

[2.0.2]

- Other Changes
  - Changed coding style of peripheral base address from "s_acmpBases" to "s_acmpBase".

[2.0.1]

- Bug Fixes
  - Fixed bug regarding the function “ACMP_SetRoundRobinConfig”. It will not continue execution but returns directly after disabling round robin mode.

CACHE64

[2.0.12]

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

[2.0.11]

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

[2.0.10]

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

[2.0.9]

- Improvements
  - Removed assert(false) in CACHE64_GetInstanceByAddr.

[2.0.8]

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

[2.0.7]

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

[2.0.6]

- Bug Fixes
  - Fixed overflow for CACHE64_GetInstanceByAddr()/CACHE64_CleanCacheByRange()/CACHE64_Invalid APIs.

[2.0.5]

- Improvement
  - Made use of FSL_FEATURE_CACHE64_CTRL_HAS_NO_WRITE_BUF feature

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

CACHE XCACHE

[2.0.4]

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

[2.0.3]

- Bug Fixes
  - Fixed CERT INT30-C violations.

[2.0.2]

- Bug Fixes
  - Updated XCACHE_InvalidateCacheByRange(), XCACHE_CleanCacheByRange(), XCACHE_CleanInvalidateCacheByRange() in case of startAddr equal to endAddr.

[2.0.1]

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

[2.0.0]

- Initial version.

CDOG

[2.1.3]

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

[2.1.2]

- Support multiple IRQs
- Fix default CONTROL values

[2.1.1]

- Remove bit CONTROL[CONTROL_CTRL].

[2.1.0]

- Rename CWT to CDOG.

[2.0.2]

- Fix MISRA-2012 issues.

[2.0.1]

- Fix doxygen issues.

[2.0.0]

- Initial version.

CLOCK

[2.5.1]

- Improvement
  - Added a check for the temporary parameter to prevent potential wrapping issues.

[2.5.0]

- New features
  - Updated CLOCK_EnableFroClkFreqCloseLoop to support timeout mechanism.
  - Added new API CLOCK_FroTuneOnce to allow frequency measurement in open loop mode.

[2.4.2]

- Improvement
  - Updated CLOCK_GetSenseAudioClkFreq for compute domain to support silicon B0.
  - Fixed MISRA issues.

[2.4.1]

- Bug fixes
  - Added kCLOCK_Pvts0.
  - Fixed bugs in CLOCK_GetWakeClk32KFreq. Reserved CLKCTL3->WAKE32KCLKDIV.

[2.4.0]

- New features
  - Added return value for CLOCK_InitMainPfd CLOCK_InitAudioPfd.
  - Updated FRO max supported frequency.
  - Added fast startup configuration for FRO.

[2.3.1]

- Bug fixes
  - Fixed bugs in CLOCK_ClearFroFlags.
  - Fixed bugs in clock_attach_id_t.

[2.3.0]

- Bug fixes
  - Added clock array for SDADC, CDOG.
  - Added FRO related APIs.

[2.2.2]

- Bug fixes
  - Fixed GPIO_CLOCKS array for GPIO_ALIAS.

[2.2.1]

- Bug fixes
  - Fixed bugs in clock_attach_id_t.
  - Fixed FRO frequency calcualtion.

[2.2.0]

- Bug fixes
  - Fixed UTICK0 clock sources.

[2.1.0]

- New features
  - Added CLOCK_GetCoreSysClkFreq API.
- Bug fixes
  - Fixed bugs in CLOCK_GetFCClkFreq and clock_attach_id_t.

[2.0.0]

- Initial version.

COMMON

[2.6.1]

- Improvements
  - Support Cortex M23.

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

- Improvements
  - Enable 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

[2.0.4]

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

[2.0.3]

- Bug fix:
  - Fix MISRA issues.

[2.0.2]

- Bug fix:
  - Fix MISRA issues.

[2.0.1]

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

[2.0.0]

- Initial version.

CTIMER

[2.3.3]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EDMA

[2.10.7]

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

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- Improvements
  - Modify the structures edma_core_mp_t, edma_core_channel_t, edma_core_tcd_t to adapt to edma5.
  - Add TCD register macro to facilitate confirmation of tcd type.
  - 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.

EZHV

[2.1.2]

- Improvements.
  - Fixed some MISRA issues.

[2.1.1]

- Improvements.
  - Move macros from source file to head file.

[2.1.0]

- Improvements.
  - Added EZHV_SetCallback function parameters.
  - Added new APIs to get EZHV core status.

[2.0.0]

- Initial version.

FLEXIO

[2.3.0]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Improvements
  - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

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

[2.1.0]

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

[2.0.4]

- Bug Fixes
  - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.2]

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

[2.0.1]

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

FLEXIO_CAMERA

[2.2.0]

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

[2.1.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.1.2]

- Improvements
  - Unified component full name to FLEXIO CAMERA (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.

[2.1.1]

- 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.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.

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

FREQME

[2.1.2]

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

[2.1.1]

- Fixed MISRA issues.

[2.1.0]

- Updated register name.

[2.0.0]

- Initial version.

GDET

[2.1.0]

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

[2.0.1]

- Fix MISRA in GDET_ReconfigureVoltageMode().

[2.0.0]

- Initial version.

GLIKEY

[2.0.1]

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

[2.0.0]

- Initial version.

GPIO

[2.8.2]

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

[2.8.1]

- Bug Fixes
  - Fixed CERT INT31-C issues.

[2.8.0]

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

[2.8.0]

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

[2.7.3]

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

[2.7.2]

- New Features
  - Support devices without PORT module.

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
  - Removed deprecated APIs.

[2.3.0]

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

[2.2.1]

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

[2.1.1]

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

[2.1.0]

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

I3C

[2.14.3]

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

[2.14.2]

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

[2.14.1]

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

[2.14.0]

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

[2.13.1]

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

[2.13.0]

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

[2.12.0]

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

[2.11.1]

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

[2.11.0]

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

[2.10.8]

- Improvements
  - Support more instances.

[2.10.7]

- Improvements
  - Fixed the potential compile warning.

[2.10.6]

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

[2.10.5]

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

[2.10.4]

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

[2.10.3]

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

[2.10.2]

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

[2.10.1]

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

[2.10.0]

- New features
  - Added I3C extended IBI data support.

[2.9.0]

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

[2.8.2]

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

[2.8.1]

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

[2.8.0]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.4]

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

[2.5.3]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH and FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND.

[2.5.2]

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

[2.5.1]

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

[2.5.0]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

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

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

I3C_EDMA

[2.2.12]

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

[2.2.11]

- Improvements
  - Fixed Coverity CERT-C violations.

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Bug Fixes
  - Fixed MISRA issue rule 11.6.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

- Improvements
  - Deleted legacy IBI data request code.

[2.1.0]

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

[2.0.1]

- Bug Fixes
  - Fixed MISRA issue rule 9.1.

[2.0.0]

- Initial version.

INPUTMUX

[2.0.10]

- Bug Fixes
  - Fixed CERT-C violations.

[2.0.9]

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

[2.0.8]

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

[2.0.7]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

- Support channel mux setting in INPUTMUX_EnableSignal().

[2.0.0]

- Initial version.

IOPCTL

[2.0.3]

- Improvement
  - Added IOPCTL_FULLDRIVE_EN for full driver enablement.
  - Fixed MISRA C-2012 10.3 issues.

[2.0.2]

- Bug fixes
  - Fix MISRA issues.

[2.0.1]

- Bug fixes
  - Remove unsupported IOPCTL_FULLDRIVE_EN(bit8) configuration.

[2.0.0]

- Initial version.

IRTC

[2.3.3]

- Bug Fixes
  - Fix CERT INT31-C issue.

[2.3.2]

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

[2.3.1]

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

[2.3.0]

- New Feature
  - Supported platforms with multiple IRTC instances.

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

- Bug Fixes
  - Fixed MISRA issues.

[2.2.0]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

ITRC

[2.4.0]

- Rework the input signal definition for better flexibility

[2.3.0]

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

[2.2.0]

- Update driver to new version and input events

[2.1.0]

- Make SYSCON glitch platform dependent.

[2.0.0]

- Initial version.

JPEGDEC

[2.0.2]

- Other Changes
  - Update driver to support more resolution due to the IP bug fix.

[2.0.1]

- Bug Fixes
  - Fixed CERT-C violations.

[2.0.0]

- Initial version.

LCDIF

[2.3.1]

- Bug Fixes
  - Fixed CERT-C violations.
  - Fixed wrong frame buffer configuration for overlay layers.

[2.3.0]

- New Features
  - Supported layer decompress mode for DC8000.

[2.2.0]

- New Features
  - Supported new layers and configurations for DC8000.
  - Added new APIs and configurations to support DBI interface.
- Bug Fixes
  - Update align calculation method, the old one can only be used when the align bytes' low bits are all zeros.

[2.1.2]

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

[2.1.1]

- Improvements
  - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.
- Bug Fixes
  - Fix MISRA-C 2012 issues.

[2.1.0]

- Bug Fixes
  - Corrected the frame buffer pixel format name.

[2.0.0]

- Initial version.

LPADC

[2.9.4]

- Improvements
  - Update LPADC_GetDefaultConfig, change default conversionAverageMode value to: kLPADC_ConversionAverage128 for 3 bit width. kLPADC_ConversionAverage1024 for 4 bit width.

[2.9.3]

- Improvements
  - Add timeout for while loop code.

[2.9.2]

- Improvements
  - Fixed CERT-C issues.

[2.9.1]

- Bug Fixes
  - Fixed incorrect channel B FIFO selection logic.

[2.9.0]

- Bug Fixes
  - Add code to handle the case where GCC[GAIN_CAL] is a signed number.
  - Split LPADC_FinishAutoCalibration function into two functions.
  - Improved LPADC driver.

[2.8.4]

- Bug Fixes
  - Remove function 'LPADC_SetOffsetValue' assert statement, this statement may cause runtime errors in existing code.

[2.8.3]

- Bug Fixes
  - Fixed SDK lpadc driver examples compile issue, move condition 'commandId < ADC_CV_COUNT' to a more appropriate location.

[2.8.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 18.1, 10.3, 10.1 and 10.4.

[2.8.1]

- Bug Fixes
  - Fixed LPADC sample mode enum name mistake.

[2.8.0]

- Improvements
  - Release peripheral from reset if necessary in init function.
- Bug Fixes
  - Fixed function LPADC_GetConvResult() issue.
  - Fixed function LPADC_SetConvCommandConfig() bugs.

[2.7.2]

- Improvements
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.1]

- Improvements
  - Corrected descriptions of several functions.
  - Improved function LPADC_GetOffsetValue and LPADC_SetOffsetValue.
  - Revert changes of feature macros for lpadc.
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.8.
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.0]

- Improvements
  - Added supports of CFG2 register.
  - Removed some useless macros.

[2.6.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules.
  - Fixed LPADC driver code compile error issue.

[2.6.1]

- Improvements
  - Updated the use of macros in the driver code.

[2.6.0]

- Improvements
  - Added the API LPADC_SetOffset12BitValue() to configure 12bit ADC conversion offset trim value manually.
  - Added the API LPADC_SetOffset16BitValue() to configure 16bit ADC conversion offset trim value manually.
  - Added API to set offset calibration mode.
  - Added configuration of alternate channel.
  - Updated auto calibration API and added calibration value conversion API.
- New feature

  - Added API LPADC_EnableHardwareTriggerCommandSelection() to enable trigger commands controlled by ADC_ETC.
  - Updated LPADC_DoAutoCalibration() to allow doing something else before the ADC initialization to be totally complete. Enhance initialization duration time of the ADC.
  - Added two new APIs to get/set calibration value.

[2.5.2]

- Improvements
  - Added while loop, LPADC_GetConvResult() will return only when the FIFO will not be empty.

[2.5.1]

- Bug Fixes
  - Fixed some typos in Lpadc driver comments.

[2.5.0]

- Improvements
  - Added missing items to enable trigger interrupts.

[2.4.0]

- New features
  - Added APIs to get/clear trigger status flags.

[2.3.0]

- Improvements
  - Removed LPADC_MeasureTemperature() function for the LPADC supports different temperature sensor calculation equations.

[2.2.1]

- Improvements
  - Optimized LPADC_MeasureTemperature() function to support the specific series with flash solidified calibration value.
  - Clean doxygen warnings.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, rule 10.8 and rule 17.7.

[2.2.0]

- New Feature
  - Added API LPADC_MeasureTemperature() to get correct temperature from the internal sensor.
- Improvements

  - Separated lpadc_conversion_resolution_mode_t with related feature macro.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.3, 10.4, 10.6, 10.7 and 17.7.

[2.1.1]

- Improvements
  - Updated the gain calibration formula.
  - Used feature to segregate the new item kLPADC_TriggerPriorityPreemptSubsequently.

[2.1.0]

- New Features
  - Added the API LPADC_SetOffsetValue() to support configure offset trim value manually.
  - Added the API LPADC_DoOffsetCalibration() to do offset calibration independently.
- Improvements
  - Improved the usage of macros and removed invalid macros.

[2.0.2]

- Improvements
  - Added support for platforms with 2 FIFOs and different calibration measures.

[2.0.1]

- Bug Fixes
  - Ensured the API LPADC_SetConvCommandConfig configure related registers correctly.

[2.0.0]

- Initial version.

LPFLEXCOMM

[2.2.1]

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

[2.2.0]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

LPI2C

[2.2.6]

- Bug Fixes
  - Fixed coverity issues.

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

- Improvements
  - Fixed doxygen warning in LPI2C_SlaveTransferHandleIRQ.

[2.2.1]

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

[2.2.0]

- Improvements
  - Support the normal LPI2C in LPFLEXCOMM driver.

[2.1.1]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.

LPI2C_EDMA

[2.0.1]

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

[2.0.0]

- Initial version.

LPSPI

[2.2.9]

- Bug Fixes
  - Fixed coverity issues.

[2.2.8]

- Bug Fixes
  - Fixed coverity issues.

[2.2.7]

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

[2.2.6]

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

[2.2.5]

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

[2.2.4]

- Improvements
  - Fixed doxygen warning in LPSPI_SlaveTransferHandleIRQ.

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

- Fixed LPSPI_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

[2.0.0]

- Initial version.

LPSPI_EDMA

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

LPUART

[2.3.4]

- Bug Fixes
  - Fixed coverity issues.

[2.3.3]

- Bug Fixes
  - Fixed coverity issues.

[2.3.2]

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

[2.3.1]

- Bug Fix
  - Fixed MISRA C-2012 violations.

[2.3.0]

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

[2.2.4]

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

[2.2.3]

- Improvements
  - Added atomic in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.2.2]

- Improvements
  - Added comment on txExtendedTimeoutValue of lpuart_timeout_config_t.

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

MIPI_DSI

[2.3.0]

- Bug Fixes
  - Fixed typo in member of dsi_transfer_t structure. The sendDscCmd and dscCmd shall be sendDcsCmd and dcsCmd.

[2.2.2]

- Bug Fixes
  - Fixed CERT-C violations.

[2.2.1]

- Bug Fixes
  - Fixed issue that VACTIVE setting shall equal to the number of active lines (height), no need to minus 1.
- Improvements
  - Update DSI_Deinit to reset peripheral.
  - Update DSI_DeinitDphy to power down DPHY using DPHY_PD_REG before powering down PLL.

[2.2.0]

- New Features
  - Added APIs to configure DBI FIFO and payload.
  - Supported new controls and configurations of DBI pixel format, PHY ready and ULPS for RT700.
  - Updated the DPI setting to use float for coefficient value for more accurate calculation.

[2.1.6]

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

[2.1.5]

- Other Changes
  - Changed to use new register naming.
  - Added workaround for Errata ERR011439. Avoid DCS long packet command writes with zero-length data payload in low-power mode, because the checksum is incorrect in this case.

[2.1.4]

- Bug Fixes
  - Fixed the MISRA issues.

[2.1.3]

- Bug Fixes
  - Fixed the DPI horizontal timing setting issue.

[2.1.2]

- Improvements
  - Supported long package read.
- Bug Fixes
  - Fixed the bug that runs to hardfault when sending long packet with 4-byte unaligned address.

[2.1.1]

- Improvements
  - Some SOC compatibility improvement.

[2.1.0]

- Improvements
  - Improved for the platforms which does not support ULPS.

[2.0.6]

- Bug Fixes
  - Fixed the timing issue that non-continuous HS clock mode does not work.

[2.0.5]

- Bug Fixes
  - Fixed kDSI_InterruptGroup1BtaTo and kDSI_InterruptGroup1HtxTo definition error.
- Improvements
  - Changed to override MIPI_DriverIRQHandler instead of MIPI_IRQHandler.

[2.0.4]

- Bug Fixes
  - Fixed MISRA C-2012 issues: 10.1, 10.3, 10.4, 10.4, 10.6, 10.7, 10.8, 11.3, 11.8, 12.2, 14.4, 16.4, 17.7.

[2.0.3]

- Improvement
  - Updated for combo phy header file.

[2.0.2]

- New Features
  - Supported sending separate DSI command from TX data array.
- Bug Fixes
  - Disabled all interrupts in DSI_Init.

[2.0.1]

- Improvements
  - Updated to support the DPHY which does not have internal DPHY PLL.

[2.0.0]

- Initial version.

MMU

[2.0.0]

- Initial version.

MRT

[2.0.5]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

MU

[2.8.1]

- Bug Fixes
  - Avoid incorrect MU_BUSY_POLL_COUNT macro use.

[2.8.0]

- New Features
  - Added MU1_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
  - Added timeout mechanism to all polling loops in MU driver code.
- Improvements
  - Updated function signatures to return status codes for better error handling:
    - Changed MU_ResetBothSides to return status_t instead of void
    - Updated MU_SendMsg to return status_t for timeout indication
    - Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
  - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.7.0]

- New Features
  - Added API MU_GetRxStatusFlags.

[2.6.0]

- New Features
  - Added API MU_GetInterruptsPending.

[2.5.1]

- Bug Fixes
  - Fixed the bug that MU_TriggerGeneralPurposeInterrupts and MU_TriggerInterrupts may trigger previous triggered general purpose interrupts again by mistake.

[2.5.0]

- New Features
  - Supported more than 4 general purpose interrupts.
  - Added seperate APIs for general purpose interrupts.

[2.4.0]

- Improvements
  - Supported the case that some features only avaiable with specific instances. These features include Hardware Reset, Boot Peer Core, Hold Reset. When using the features with instances which don't support them, driver will report error.

[2.3.3]

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

[2.3.2]

- Improvements
  - Supported platforms which don't have CCR0[RSTH], CCR0[CLKE], CCR0[HR], CCR0[HRM].

[2.3.1]

- Bug Fixes
  - Fixed build error for platforms which have CCR0[RSTH], but no CCR0[NMI].

[2.3.0]

- New features
  - Added support for i.MX RT7xx.

[2.2.1]

- Bug Fixes
  - Fixed issue that MU_GetInstance() is defined but never used.

[2.2.0]

- New features
  - Added support for i.MX RT118x.
- Bug Fixes
  - Fixed general purpose interrupt bug.
- Other Changes
  - Change _mu_interrupt_trigger item value.

[2.1.2]

- Bug Fixes
  - Fixed bug that general purpose interrupt can't be configured.

[2.1.1]

- Bug Fixes
  - Fixed MISRA C-2012 issues.

[2.1.0]

- Improvements
  - Added new enum mu_msg_reg_index_t.

[2.0.0]

- Initial version.

OSTIMER

[2.2.6]

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

[2.2.5]

- Improvements
  - Support binary encoded ostimer.

[2.2.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.2.3]

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

[2.2.2]

- Improvements
  - Support devices with different OSTIMER instance name.

[2.2.1]

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

[2.2.0]

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

[2.1.2]

- Bug Fixes
  - Fixed MISRA-2012 rule 10.8.

[2.1.1]

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

[2.1.0]

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

[2.0.3]

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

[2.0.2]

- Improvements
  - Added support for OSTIMER0

[2.0.1]

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

[2.0.0]

- Initial version.

PDM

[2.9.3]

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

[2.9.2]

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

[2.9.1]

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

[2.9.0]

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

[2.8.1]

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

[2.8.0]

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

[2.7.4]

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

[2.7.3]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_VADEF to remove the support of VADEF bit-field in VAD0_STAT register.

[2.7.2]

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

[2.7.1]

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

[2.7.0]

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

[2.6.0]

- Improvements
  - Added new features FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ/FSL_FEATURE_PDM_HAS_DC_OUT

[2.5.0]

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

[2.4.1]

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

[2.4.0]

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

[2.3.0]

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

[2.2.1]

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

[2.2.0]

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

[2.1.1]

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

[2.1.0]

- Improvements
  - Added software buffer queue for transactional API.

[2.0.1]

- Improvements
  - Improved HWVAD feature.

[2.0.0]

- Initial version.

PDM_EDMA

[2.6.5]

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

[2.6.4]

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

[2.6.3]

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

[2.6.2]

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

[2.6.1]

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

[2.6.0]

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

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.5.0

PINT

[2.2.0]

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

[2.1.13]

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

[2.1.12]

- Bug Fixes
  - Fixed coverity issue.

[2.1.11]

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

[2.1.10]

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

[2.1.9]

- Bug Fixes
  - Fixed MISRA-2012 rule 8.4.

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

PNGDEC

[2.0.1]

- Bug Fixes
  - Fixed CERT-C violations.

[2.0.0]

- Initial version.

POWER

[2.5.0]

- New features
  - Added API POWER_EnableSleepAFBB to configure body bias to AFBB in deep sleep mode.
  - Added user configurable macro POWER_LDO_SAFE_MARGIN(x) with default value “0” to allow adding LDO output safe margin when using API POWER_ConfigRegulatorSetpointsForFreq.

[2.4.2]

- Bug fixes
  - Clear PDRUNCFG0 bit6 and bit8 before entering Deep Power Down(DPD) mode to make sure the domain is reset correctly after wake up from DPD.

[2.4.1]

- Bug fixes
  - Fixed bugs in POWER_DisableSleepRegulators POWER_EnableSleepRegulators.

[2.4.0]

- New features
  - Added APIs POWER_CalcVoltLevel, POWER_ConfigRegulatorSetpointsForFreq.

[2.3.0]

- Bug fixes
  - Fixed bugs in LDO Setpoint setting APIs.
  - Fixed bugs in XSPI recovery when XIP.
- New features
  - Added API POWER_EnterPowerMode.

[2.2.1]

- Bug fixes
  - Fixed bugs in enum _power_hwwake_src.

[2.2.0]

- New features
  - Added parameter checks in setpoint configuration APIs.
- Bug fixes
  - Fixed bugs in power mode entry API.

[2.1.0]

- New features
  - Added POWER_ConfigLvdSetpoints API.

[2.0.0]

- Initial version.

PUFv3

[2.0.4]

- Fix warning.

[2.0.3]

- Update for various PUF CTRL wrapper.

[2.0.2]

- Fix MISRA issue in driver.

[2.0.1]

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

[2.0.0]

- Initial version.

PVTS

[2.0.2]

- New features
- Updated driver per ROMAPI changes.

[2.0.1]

- Improvement
  - Fixed MSG violations. Fixed MISRA C-2012 violations.

[2.0.0]

- Initial version.

RESET

[2.1.0]

- New features
  - Added RESET_SetFlashStateContext API.

[2.0.2]

- Bug fixes
  - Added FREQME_RSTS_N.

[2.0.1]

- Refined the GPIO_RSTS array so that the element number matches the instance index
- Added SDADC_RSTS.

[2.0.0]

- initial version.

ROMAPI

[2.1.0]

- New features
  - Updated OTP API to support Silicon B0.

[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

SCTIMER

[2.5.1]

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

[2.5.0]

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

[2.4.9]

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

[2.4.8]

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

[2.4.7]

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

[2.4.6]

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

[2.4.5]

- Bug Fixes
  - Fix SCT_EV_STATE_STATEMSKn macro build error.

[2.4.4]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

[2.3.0]

- Bug Fixes
  - Fixed the potential overflow issue of pulseperiod variable in SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle API.
  - Fixed the issue of SCTIMER_CreateAndScheduleEvent API does not correctly work with 32 bit unified counter.
  - Fixed the issue of position of clear counter operation in SCTIMER_Init API.
- Improvements
  - Update SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle to support generate 0% and 100% PWM signal.
  - Add SCTIMER_SetupEventActiveDirection API to configure event activity direction.

  - Update SCTIMER_StartTimer/SCTIMER_StopTimer API to support start/stop low counter and high counter at the same time.
  - Add SCTIMER_SetCounterState/SCTIMER_GetCounterState API to write/read counter current state value.
  - Update APIs to make it meaningful.
    * SCTIMER_SetEventInState
    * SCTIMER_ClearEventInState
    * SCTIMER_GetEventInState

[2.2.0]

- Improvements
  - Updated for 16-bit register access.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.1]

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

[2.0.0]

- Initial version.

SDADC

[2.0.1]

- Improvements
  - Fixed MISRA C-2012 issues.

[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.

SYSPM

[2.3.1]

- Improvements
  - Add timeout for while loop.

[2.3.0]

- New Features
  - Supported instruction counter.

[2.2.0]

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

[2.1.0]

- New Features
  - Added new APIs SYSPM_Init and SYSPM_Deinit.

[2.0.0]

- Initial version.

TRNG

[2.0.19]

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

[2.0.18]

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

[2.0.17]

- New features:
  - Add support for RT700.

[2.0.16]

- Improvements:
  - Added support for Dual oscillator mode.

[2.0.15]

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

[2.0.14]

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

[2.0.13]

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

[2.0.12]

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

[2.0.11]

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

[2.0.10]

- Bug Fix:
  - Fixed doxygen issues.

[2.0.9]

- Bug Fix:
  - Fix HIS_CCM metrics issues.

[2.0.8]

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

[2.0.7]

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

[2.0.6]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

- Improvements:
  - fix MISRA issues
    - Rule 14.4.

[2.0.1]

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

[2.0.0]

- Initial version.

USDHC

[2.8.6]

- Bug Fixes
  - Invalidate cache after blocking read.

[2.8.5]

- Improvements
  - Enable the driver to be AARCH64 compatible.

[2.8.4]

- Improvements
  - Add feature macro FSL_FEATURE_USDHC_HAS_NO_VS18.

[2.8.3]

- Improvements
  - Improved api USDHC_EnableAutoTuningForCmdAndData to adapt to new bit field name for USDHC_VEND_SPEC2 register.

[2.8.2]

- Improvements
  - Added feature macro FSL_FEATURE_USDHC_HAS_NO_VOLTAGE_SELECT.

[2.8.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9.

[2.8.0]

- Improvements
  - Fixed the mmc boot transfer failed issue which is caused by the Dma complete interrupt not enabled.
  - Marked api USDHC_AdjustDelayForManualTuning as deprecated and added new api USDHC_SetTuingDelay/USDHC_GetTuningDelayStatus.
  - Improved the manual tuning flow accroding to specification.
  - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.7.0]

- Improvements
  - Added api USDHC_TransferScatterGatherADMANonBlocking to support scatter gather transfer.
  - Added feature FSL_FEATURE_USDHC_REGISTER_HOST_CTRL_CAP_HAS_NO_RETUNING_TIME_COUNT for re-tuning time counter field in HOST_CTRL_CAP register.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 10.1, 10.3, 10.4, 8.4.

[2.6.0]

- Improvements
  - Added api USDHC_SetStandardTuningCounter to support adjust tuning counter of Standard tuning.

[2.5.1]

- Improvements
  - Used different status code for command and data interrupt callback.
  - Added cache line invalidate for receive buffer in driver IRQ handler to fix CM7 speculative access issue.

[2.5.0]

- Improvements
  - Added new api USDHC_SetStrobeDllOverride for HS400 strobe dll override mode delay taps configurations.
  - Corrected the STROBE DLL configurations sequence.

[2.4.0]

- Improvements
  - Added feature macro for read/write burst length.
    - Disabled redundant interrupt per different transfer request.
    - Disabled interrupt and reset command/data pointer in handle when transfer completes.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 15.7, 4.7, 16.4, 10.1, 10.3, 10.4, 11.3, 14.4, 10.6, 17.7, 16.1, 16.3.
  - Fixed PA082 build warning.
  - Fixed logically dead code Coverity issue.

[2.3.0]

- Improvements
  - Added USDHC_SetDataConfig API to support manual tuning.
  - Removed the limitaion that source clock must be bigger than the target in function USDHC_SetSdClock by using source clock frequency as target directly.
  - Added peripheral reset in USDHC_Init function.
  - Added tuning reset support in function USDHC_Reset function.

[2.2.8]

- Bug Fixes
  - Fixed out-of bounds write in function USDHC_ReceiveCommandResponse.

[2.2.7]

- Improvements
  - Added API USDHC_GetEnabledInterruptStatusFlags and used in USDHC_TransferHandleIRQ.
  - Removed useless member interruptFlags in usdhc_handle_t.

[2.2.6]

- Improvements
  - Added address align check for ADMA descriptor table address.
  - Changed USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY to (65536-4096) to make sure the data address is 4KB align for a transfer which need more than one ADMA1 descriptor.

[2.2.5]

- Bug Fixes
  - Fixed MDK 66-D warning.

[2.2.4]

- Bug Fixes
  - Fixed issue that real clock frequency wss mismatched with target clock frequency, which was caused by an incorrect prescaler calculation.
- New Features
  - Added control macro to enable/disable the CLOCK code in current driver.

[2.2.3]

- Bug Fixes
  - Fixed issue where AMDA did not disable with DMAEN clear.
- Improvements
  - Improved set clock function to check the output frequency range.
  - Dynamic set SDCLKFS during DDR enable or disable.

[2.2.2]

- Improvements
  - Improved read transfer cache maintain operation, combined clean, and invalidated them into one function.

[2.2.1]

- Bug Fixes
  - Disabled the invalidate cache operation for tuning.

[2.2.0]

- Improvements
  - Improved USDHC to support MMC boot feature.

[2.1.3]

- Bug Fixes
  - Fixed MISRA issue.

[2.1.2]

- Bug Fixes
  - Fixed Coverity issue.
  - Added base address and userData parameter for all callback functions.

[2.1.1]

- Improvements
  - Added cache maintain operation.
  - Added timeout status check for the DATA transfer which ignore error.
  - Added feature macro for SDR50/SDR104 mode.
  - Removed useless IRQ handler from different platforms.

[2.1.0]

- Improvements
  - Integrated tuning into transfer function.
  - Added strobe DLL feature.
  - Added enableAutoCommand23 in data structure.
  - Removed enable card clock function because the controller would handle the clock on/off.

[2.0.0]

- Initial version.

UTICK

[2.0.5]

- Improvements
  - Improved for SOC RW610.

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

WWDT

[2.1.10]

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

- Bug Fixes
  - Fixed the issue that the watchdog reset event affected the system from PMC.
  - Fixed the issue of setting watchdog WDPROTECT field without considering the backwards compatibility.
  - Fixed the issue of clearing bit fields by mistake in the function of WWDT_ClearStatusFlags.

[2.1.5]

- Bug Fixes
  - deprecated a unusable API in WWWDT driver.
    - WWDT_Disable

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.0]

- Initial version.

XSPI

[2.5.2]

- Improvements
  - Fixed an issue with incorrect cache range address calculation. It caused issue when the cache operation size is equal to a multiple of the cache line size + 1.

[2.5.1]

- Improvements
  - Updated default value of sfpArbitrationLockTimeoutValue and ipAccessTimeoutValue to 0xFFFFFFFFUL.
  - Added #if defined(CACHE64_CTRL0_BASE) ... #endif section to support some devices that do not support CACHE64.

[2.5.0]

- Improvements
  - Updated XSPI_TransferBlocking() to support use case the transfersize bigger than page size.
  - Updated xspi_device_interface_type_t to support changes of page size for hyperram interface.
  - Updated XSPI_ReadBlocking() to fix an potential issue which cause TIMEOUT error.

[2.4.0]

- New Features
  - Added functions of control cache64.

[2.3.0]

- Improvements
  - Added new interface: XSPI_StartIpAccessNonBlocking();
  - Provied driver Irq handler for xspi driver;
  - Use ERRSTAT[ARB_WIN] to replace FSMSTAT[VLD].

[2.2.1]

- Improvements
  - Moved some frequently used variables(s_tgSfarsRegOffset, s_tgIpcrsRegOffset, s_sfpTgIpcrRegOffset, s_sfpTgIpSfarRegOffset, s_tgMdadRegOffset) to common code to offload of stack.
  - Added return status in case of timeout flag asserted during IP read access.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rules.

[2.2.0]

- Improvements
  - Improved xspi_device_config_t structure, removed some not-device related members.
  - Improved xspi_config_t structure, removed some device related settings.
  - Renamed XSPI_SetFlashConfig() to XSPI_SetDeviceConfig().
  - Decoupled settings of IP access, AHB access.
  - Added low-level interfaces to support IP access(including SFP), and AHB access(including performance monitor).

[2.1.0]

- Bug Fixes
  - Fixed issue of XSPI_Init() function, set MCR[MDIS] to disable clocks before settings of MCR register.
  - Unused deviceConfig in XSPI_GetDefaultConfig() function.

- Improvements
  - Updated structure for dllconfig, added method to allow user custom dll parameters.
  - Updated XSPI_UpdateDllValue() function to align with changes of dllconfig.
  - Move device specific value to device feature file.

[2.0.1]

- Bug Fixes
  - Fixed the XSPI DLL function.
  - Added ALIGN and X16Mode macro definition in driver.
  - updated XSPI PSRAM example.

[2.0.0]

- Initial version.

XSPI EDMA Driver

[2.0.2]

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

[2.0.1]

- Improvements
  - Invoked new defined interface of xspi driver.
  - In callback function, disable RX/TX DMA.

[2.0.0]

- Initial version.

1.7 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.

MIMXRT798S

1.8 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.

1.8.1 VG-Lite GPU Library

vglite

1.8.2 Multicore

multicore

1.8.3 MCU Boot

mcuboot_opensource

1.8.4 eIQ

eIQ

1.8.5 FreeMASTER

freemaster

1.8.6 AWS IoT

aws_iot

1.8.7 NXP Wi-Fi

wifi-bluetooth-802.15.4

1.8.8 FreeRTOS

FreeRTOS

1.8.9 Wireless EdgeFast Bluetooth PAL

EdgeFast Bluetooth

1.8.10 lwIP

lwip

1.8.11 File systemFatfs

fatfs

1.8.12 DSP Audio Streamer

multicore-xaf

Chapter 2

MIMXRT798S

2.1 ACMP: Analog Comparator Driver

void ACMP_Init(CMP_Type *base, const *acmp_config_t* *config)

Initializes the ACMP.

The default configuration can be got by calling ACMP_GetDefaultConfig().

Parameters

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

void ACMP_Deinit(CMP_Type *base)

Deinitializes the ACMP.

Parameters

- base – ACMP peripheral base address.

void ACMP_GetDefaultConfig(*acmp_config_t* *config)

Gets the default configuration for ACMP.

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

Example:

```
config->enableHighSpeed = false;
config->enableInvertOutput = false;
config->useUnfilteredOutput = false;
config->enablePinOut = false;
config->enableHysteresisBothDirections = false;
config->hysteresisMode = kACMP_hysteresisMode0;
```

Parameters

- config – Pointer to ACMP configuration structure.

void ACMP_Enable(CMP_Type *base, bool enable)

Enables or disables the ACMP.

Parameters

- base – ACMP peripheral base address.
- enable – True to enable the ACMP.

void ACMP_EnableLinkToDAC(CMP_Type *base, bool enable)

Enables the link from CMP to DAC enable.

When this bit is set, the DAC enable/disable is controlled by the bit CMP_C0[EN] instead of CMP_C1[DACEN].

Parameters

- base – ACMP peripheral base address.
- enable – Enable the feature or not.

void ACMP_SetChannelConfig(CMP_Type *base, const *acmp_channel_config_t* *config)

Sets the channel configuration.

Note that the plus/minus mux's setting is only valid when the positive/negative port's input isn't from DAC but from channel mux.

Example:

```
acmp_channel_config_t configStruct = {0};
configStruct.positivePortInput = kACMP_PortInputFromDAC;
configStruct.negativePortInput = kACMP_PortInputFromMux;
configStruct.minusMuxInput = 1U;
ACMP_SetChannelConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.
- config – Pointer to channel configuration structure.

void ACMP_EnableDMA(CMP_Type *base, bool enable)

Enables or disables DMA.

Parameters

- base – ACMP peripheral base address.
- enable – True to enable DMA.

void ACMP_SetFilterConfig(CMP_Type *base, const *acmp_filter_config_t* *config)

Configures the filter.

The filter can be enabled when the filter count is bigger than 1, the filter period is greater than 0 and the sample clock is from divided bus clock or the filter is bigger than 1 and the sample clock is from external clock. Detailed usage can be got from the reference manual.

Example:

```
acmp_filter_config_t configStruct = {0};
configStruct.filterCount = 5U;
configStruct.filterPeriod = 200U;
configStruct.enableSample = false;
ACMP_SetFilterConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.
- config – Pointer to filter configuration structure.

void ACMP_SetDACConfig(CMP_Type *base, const *acmp_dac_config_t* *config)

Configures the internal DAC.

Example:

```
acmp_dac_config_t configStruct = {0};
configStruct.referenceVoltageSource = kACMP_VrefSourceVin1;
configStruct.DACValue = 20U;
configStruct.enableOutput = false;
configStruct.workMode = kACMP_DACWorkLowSpeedMode;
ACMP_SetDACConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.
- config – Pointer to DAC configuration structure. "NULL" is for disabling the feature.

void ACMP_EnableInterrupts(CMP_Type *base, uint32_t mask)

Enables interrupts.

Parameters

- base – ACMP peripheral base address.
- mask – Interrupts mask. See "_acmp_interrupt_enable".

void ACMP_DisableInterrupts(CMP_Type *base, uint32_t mask)

Disables interrupts.

Parameters

- base – ACMP peripheral base address.
- mask – Interrupts mask. See "_acmp_interrupt_enable".

uint32_t ACMP_GetStatusFlags(CMP_Type *base)

Gets status flags.

Parameters

- base – ACMP peripheral base address.

Returns

Status flags asserted mask. See "_acmp_status_flags".

void ACMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)

Clears status flags.

Parameters

- base – ACMP peripheral base address.
- mask – Status flags mask. See "_acmp_status_flags".

void ACMP_SetDiscreteModeConfig(CMP_Type *base, const *acmp_discrete_mode_config_t* *config)

Configure the discrete mode.

Configure the discrete mode when supporting 3V domain with 1.8V core.

Parameters

- base – ACMP peripheral base address.
- config – Pointer to configuration structure. See "acmp_discrete_mode_config_t".

void ACMP_GetDefaultDiscreteModeConfig(*acmp_discrete_mode_config_t* *config)

Get the default configuration for discrete mode setting.

Parameters

- config – Pointer to configuration structure to be restored with the setting values.

FSL_ACMP_DRIVER_VERSION

ACMP driver version 2.4.0.

enum _acmp_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kACMP_OutputRisingInterruptEnable

Enable the interrupt when comparator outputs rising.

enumerator kACMP_OutputFallingInterruptEnable

Enable the interrupt when comparator outputs falling.

enum _acmp_status_flags

Status flag mask.

Values:

enumerator kACMP_OutputRisingEventFlag

Rising-edge on compare output has occurred.

enumerator kACMP_OutputFallingEventFlag

Falling-edge on compare output has occurred.

enumerator kACMP_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum _acmp_offset_mode

Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by acmp_hysteresis_mode_t is valid for both directions.

Values:

enumerator kACMP_OffsetLevel0

The comparator hard block output has level 0 offset internally.

enumerator kACMP_OffsetLevel1

The comparator hard block output has level 1 offset internally.

enum _acmp_hysteresis_mode

Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

Values:

enumerator kACMP_HysteresisLevel0

Offset is level 0 and Hysteresis is level 0.

enumerator kACMP_HysteresisLevel1

Offset is level 0 and Hysteresis is level 1.

enumerator kACMP_HysteresisLevel2

Offset is level 0 and Hysteresis is level 2.

enumerator kACMP_HysteresisLevel3
Offset is level 0 and Hysteresis is level 3.

enum _acmp_reference_voltage_source
CMP Voltage Reference source.

Values:

enumerator kACMP_VrefSourceVin1
Vin1 is selected as resistor ladder network supply reference Vin.

enumerator kACMP_VrefSourceVin2
Vin2 is selected as resistor ladder network supply reference Vin.

enum _acmp_port_input
Port input source.

Values:

enumerator kACMP_PortInputFromDAC
Port input from the 8-bit DAC output.

enumerator kACMP_PortInputFromMux
Port input from the analog 8-1 mux.

enum _acmp_dac_work_mode
Internal DAC's work mode.

Values:

enumerator kACMP_DACWorkLowSpeedMode
DAC is selected to work in low speed and low power mode.

enumerator kACMP_DACWorkHighSpeedMode
DAC is selected to work in high speed high power mode.

typedef enum *_acmp_offset_mode* acmp_offset_mode_t
Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by acmp_hysteresis_mode_t is valid for both directions.

typedef enum *_acmp_hysteresis_mode* acmp_hysteresis_mode_t
Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

typedef enum *_acmp_reference_voltage_source* acmp_reference_voltage_source_t
CMP Voltage Reference source.

typedef enum *_acmp_port_input* acmp_port_input_t
Port input source.

typedef enum *_acmp_dac_work_mode* acmp_dac_work_mode_t
Internal DAC's work mode.

typedef struct *_acmp_config* acmp_config_t
Configuration for ACMP.

typedef struct *_acmp_channel_config* acmp_channel_config_t

Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

typedef struct *_acmp_filter_config* acmp_filter_config_t

Configuration for filter.

typedef struct *_acmp_dac_config* acmp_dac_config_t

Configuration for DAC.

typedef struct *_acmp_discrete_mode_config* acmp_discrete_mode_config_t

Configuration for discrete mode.

CMP_C0_CFx_MASK

The mask of status flags cleared by writing 1.

struct _acmp_config

#include <fsl_acmp.h> Configuration for ACMP.

Public Members

acmp_offset_mode_t offsetMode

Offset mode.

acmp_hysteresis_mode_t hysteresisMode

Hysteresis mode.

bool enableHighSpeed

Enable High Speed (HS) comparison mode.

bool enableInvertOutput

Enable inverted comparator output.

bool useUnfilteredOutput

Set compare output(COUT) to equal COUTA(true) or COUT(false).

bool enablePinOut

The comparator output is available on the associated pin.

struct _acmp_channel_config

#include <fsl_acmp.h> Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

Public Members

acmp_port_input_t positivePortInput

Input source of the comparator's positive port.

uint32_t plusMuxInput

Plus mux input channel(0~7).

acmp_port_input_t negativePortInput

Input source of the comparator's negative port.

uint32_t minusMuxInput

Minus mux input channel(0~7).

struct _acmp_filter_config

#include <fsl_acmp.h> Configuration for filter.

Public Members

uint32_t filterCount

Filter Sample Count. Available range is 1-7, 0 would cause the filter disabled.

uint32_t filterPeriod

Filter Sample Period. The divider to bus clock. Available range is 0-255.

struct _acmp_dac_config

#include <fsl_acmp.h> Configuration for DAC.

Public Members

acmp_reference_voltage_source_t referenceVoltageSource

Supply voltage reference source.

uint32_t DACValue

Value for DAC Output Voltage. Available range is 0-255.

bool enableOutput

Enable the DAC output.

struct _acmp_discrete_mode_config

#include <fsl_acmp.h> Configuration for discrete mode.

Public Members

bool enablePositiveChannelDiscreteMode

Positive Channel Continuous Mode Enable. By default, the continuous mode is used.

bool enableNegativeChannelDiscreteMode

Negative Channel Continuous Mode Enable. By default, the continuous mode is used.

2.2 CACHE: CACHE Memory Controller

uint32_t CACHE64_GetInstance(CACHE64_POLSEL_Type *base)

Returns an instance number given peripheral base address.

Parameters

- base – The peripheral base address.

Returns

CACHE64_POLSEL instance number starting from 0.

uint32_t CACHE64_GetInstanceByAddr(uint32_t address)

brief Returns an instance number given physical memory address.

param address The physical memory address.

Returns

CACHE64_CTRL instance number starting from 0.

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

Initializes an CACHE64 instance with the user configuration structure.

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

Parameters

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

Return values

kStatus_Success – CACHE64 initialize succeed

void CACHE64_GetDefaultConfig(*cache64_config_t* *config)

Gets the default configuration structure.

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

Parameters

- config – Pointer to a configuration structure.

void CACHE64_EnableCache(CACHE64_CTRL_Type *base)

Enables the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_DisableCache(CACHE64_CTRL_Type *base)

Disables the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_InvalidateCache(CACHE64_CTRL_Type *base)

Invalidates the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_InvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cache by range.

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

Parameters

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

void CACHE64_CleanCache(CACHE64_CTRL_Type *base)

Cleans the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cache by range.

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

Parameters

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

void CACHE64_CleanInvalidateCache(CACHE64_CTRL_Type *base)

Cleans and invalidates the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate cache by range.

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

Parameters

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

void CACHE64_EnableWriteBuffer(CACHE64_CTRL_Type *base, bool enable)

Enables/disables the write buffer.

Parameters

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

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

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

Parameters

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

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

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

Parameters

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

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

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

Parameters

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

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

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

Parameters

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

FSL_CACHE_DRIVER_VERSION

cache driver version.

enum _cache64_policy

Level 2 cache controller way size.

Values:

enumerator kCACHE64_PolicyNonCacheable

Non-cacheable

enumerator kCACHE64_PolicyWriteThrough
Write through

enumerator kCACHE64_PolicyWriteBack
Write back

typedef enum *_cache64_policy* cache64_policy_t
Level 2 cache controller way size.

typedef struct *_cache64_config* cache64_config_t
CACHE64 configuration structure.

CACHE64_LINESIZE_BYTE
cache line size.

CACHE64_REGION_NUM
cache region number.

CACHE64_REGION_ALIGNMENT
cache region alignment.

struct _cache64_config
#include <fsl_cache.h> CACHE64 configuration structure.

Public Members

uint32_t boundaryAddr[(3U) - 1]
< The cache controller can divide whole memory into 3 regions. Boundary address is the FlexSPI internal address (start from 0) instead of system address (start from FlexSPI AMBA base) to split adjacent regions and must be 1KB aligned. The boundary address itself locates in upper region. Cacheable policy for each region.

2.3 CDOG

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

This function initializes CDOG block and setting.

Parameters

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

Returns
Status of the init operation

void CDOG_Deinit(CDOG_Type *base)
Deinitialize CDOG.

This function deinitializes CDOG secure counter.

Parameters

- base – CDOG peripheral base address

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

This function initialize CDOG config structure to default values.

Parameters

- conf – CDOG configuration structure

void CDOG_Stop(CDOG_Type *base, uint32_t stop)

Stops secure counter and instruction timer.

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

Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter

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

Sets secure counter and instruction timer values.

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

Parameters

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

void CDOG_Check(CDOG_Type *base, uint32_t check)

Checks secure counter.

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

Parameters

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

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

Sets secure counter and instruction timer values.

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

Parameters

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

void CDOG_Add(CDOG_Type *base, uint32_t add)

Add value to secure counter.

This function add specified value to secure counter.

Parameters

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

void CDOG_Add1(CDOG_Type *base)

Add 1 to secure counter.

This function add 1 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add16(CDOG_Type *base)

Add 16 to secure counter.

This function add 16 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add256(CDOG_Type *base)

Add 256 to secure counter.

This function add 256 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub(CDOG_Type *base, uint32_t sub)

brief Substract value to secure counter

This function substract specified value to secure counter.

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

void CDOG_Sub1(CDOG_Type *base)

Substract 1 from secure counter.

This function substract specified 1 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub16(CDOG_Type *base)

Substract 16 from secure counter.

This function substract specified 16 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub256(CDOG_Type *base)

Substract 256 from secure counter.

This function substract specified 256 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_WritePersistent(CDOG_Type *base, uint32_t value)

Set the CDOG persistent word.

Parameters

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

uint32_t CDOG_ReadPersistent(CDOG_Type *base)

Get the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.

Returns

The persistent word.

FSL_CDOG_DRIVER_VERSION

Defines CDOG driver version 2.1.3.

Change log:

- Version 2.1.3
  - Re-design multiple instance IRQs and Clocks
  - Add fix for RESTART command errata
- Version 2.1.2
  - Support multiple IRQs
  - Fix default CONTROL values
- Version 2.1.1
  - Remove bit CONTROL[CONTROL_CTRL]
- Version 2.1.0
  - Rename CWT to CDOG
- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - Fix doxygen issues
- Version 2.0.0
  - initial version

enum ___cdog_debug_Action_ctrl_enum

Values:

enumerator kCDOG_DebugHaltCtrl_Run

enumerator kCDOG_DebugHaltCtrl_Pause

enum ___cdog_irq_pause_ctrl_enum

Values:

enumerator kCDOG_IrqPauseCtrl_Run

enumerator kCDOG_IrqPauseCtrl_Pause

enum ___cdog_fault_ctrl_enum

Values:

enumerator kCDOG_FaultCtrl_EnableReset

enumerator kCDOG_FaultCtrl_EnableInterrupt

enumerator kCDOG_FaultCtrl_NoAction

enum ___code_lock_ctrl_enum

Values:

enumerator kCDOG_LockCtrl_Lock

enumerator kCDOG_LockCtrl_Unlock

typedef uint32_t secure_counter_t

SC_ADD(add)

SC_ADD1

SC_ADD16

SC_ADD256

SC_SUB(sub)

SC_SUB1

SC_SUB16

SC_SUB256

SC_CHECK(val)

struct cdog_config_t

#include <fsl_cdog.h>

2.4 Clock Driver

enum _clock_usb_src

USB clock source definition.

Values:

enumerator kCLOCK_Usb480M

Use 480M.

enumerator kCLOCK_UsbSrcUnused

Used when the function does not care the clock source.

enum _clock_usb_phy_src

Source of the USB HS PHY.

Values:

enumerator kCLOCK_Usbphy480M

Use 480M.

enum _clock_ip_name

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

enumerator kCLOCK_IpInvalid

Invalid Ip Name.

enumerator kCLOCK_Xcache1

Clock gate name: Code cache

enumerator kCLOCK_Xcache0
Clock gate name: System cache

enumerator kCLOCK_Ocotp0
Clock gate name: VDD2 OTP0

enumerator kCLOCK_Sleepcon0
Clock gate name: SLEEPCON_CMPT

enumerator kCLOCK_Syscon0
Clock gate name: SYSCON_CMPT

enumerator kCLOCK_Glikey0
Clock gate name: GLIKEY0

enumerator kCLOCK_Glikey3
Clock gate name: GLIKEY3

enumerator kCLOCK_TpiuTraceClkin
Clock gate name: TPIU_TRACECLKIN

enumerator kCLOCK_SWOTraceClkin
Clock gate name: SWO_TRACECLKIN

enumerator kCLOCK_Tsclk
Clock gate name: TRACE

enumerator kCLOCK_Dma0
Clock gate name: DMA0

enumerator kCLOCK_Dma1
Clock gate name: DMA1

enumerator kCLOCK_PkcRam
Clock gate name: PKC RAM

enumerator kCLOCK_Pkc
Clock gate name: PKC

enumerator kCLOCK_Romcp
Clock gate name: ROMCP

enumerator kCLOCK_Xspi0
Clock gate name: XSPI0

enumerator kCLOCK_Xspi1
Clock gate name: XSPI1

enumerator kCLOCK_Cache64ctrl0
Clock gate name: CACHE64_0

enumerator kCLOCK_Cache64ctrl1
Clock gate name: CACHE64_1

enumerator kCLOCK_Puf
Clock gate name: QK_SUBSYS

enumerator kCLOCK_Mmu0
Clock gate name: MMU0

enumerator kCLOCK_Mmu1
Clock gate name: MMU1

enumerator kCLOCK_Gpio0
Clock gate name: GPIO0

enumerator kCLOCK_Gpio1
Clock gate name: GPIO1

enumerator kCLOCK_Gpio2
Clock gate name: GPIO2

enumerator kCLOCK_Gpio3
Clock gate name: GPIO3

enumerator kCLOCK_Gpio4
Clock gate name: GPIO4

enumerator kCLOCK_Gpio5
Clock gate name: GPIO5

enumerator kCLOCK_Gpio6
Clock gate name: GPIO6

enumerator kCLOCK_Gpio7
Clock gate name: GPIO7

enumerator kCLOCK_Sct0
Clock gate name: SCT0

enumerator kCLOCK_Cdog0
Clock gate name: CDOG0

enumerator kCLOCK_Cdog1
Clock gate name: CDOG1

enumerator kCLOCK_Cdog2
Clock gate name: CDOG2

enumerator kCLOCK_LPFlexComm0
Clock gate name: LP_Flexcomm0

enumerator kCLOCK_LPFlexComm1
Clock gate name: LP_Flexcomm1

enumerator kCLOCK_LPUart0
Clock gate name: LPUART0

enumerator kCLOCK_LPUart1
Clock gate name: LPUART1

enumerator kCLOCK_LPI2c0
Clock gate name: LPI2C0

enumerator kCLOCK_LPI2c1
Clock gate name: LPI2C1

enumerator kCLOCK_LPSpi0
Clock gate name: LPSPI0

enumerator kCLOCK_LPSpi1
Clock gate name: LPSPI1

enumerator kCLOCK_LPFlexComm2
Clock gate name: LP_Flexcomm2

enumerator kCLOCK_LPFlexComm3
Clock gate name: LP_Flexcomm3

enumerator kCLOCK_LPFlexComm4
Clock gate name: LP_Flexcomm4

enumerator kCLOCK_LPFlexComm5
Clock gate name: LP_Flexcomm5

enumerator kCLOCK_LPFlexComm6
Clock gate name: LP_Flexcomm6

enumerator kCLOCK_LPFlexComm7
Clock gate name: LP_Flexcomm7

enumerator kCLOCK_LPFlexComm8
Clock gate name: LP_Flexcomm8

enumerator kCLOCK_LPFlexComm9
Clock gate name: LP_Flexcomm9

enumerator kCLOCK_LPFlexComm10
Clock gate name: LP_Flexcomm10

enumerator kCLOCK_LPFlexComm11
Clock gate name: LP_Flexcomm11

enumerator kCLOCK_LPFlexComm12
Clock gate name: LP_Flexcomm12

enumerator kCLOCK_LPFlexComm13
Clock gate name: LP_Flexcomm13

enumerator kCLOCK_LPUart2
Clock gate name: LPUART2

enumerator kCLOCK_LPUart3
Clock gate name: LPUART3

enumerator kCLOCK_LPUart4
Clock gate name: LPUART4

enumerator kCLOCK_LPUart5
Clock gate name: LPUART5

enumerator kCLOCK_LPUart6
Clock gate name: LPUART6

enumerator kCLOCK_LPUart7
Clock gate name: LPUART7

enumerator kCLOCK_LPUart8
Clock gate name: LPUART8

enumerator kCLOCK_LPUart9
Clock gate name: LPUART9

enumerator kCLOCK_LPUart10
Clock gate name: LPUART10

enumerator kCLOCK_LPUart11
Clock gate name: LPUART11

enumerator kCLOCK_LPUart12
Clock gate name: LPUART12

enumerator kCLOCK_LPUart13
Clock gate name: LPUART13

enumerator kCLOCK_LPI2c2
Clock gate name: LPI2C2

enumerator kCLOCK_LPI2c3
Clock gate name: LPI2C3

enumerator kCLOCK_LPI2c4
Clock gate name: LPI2C4

enumerator kCLOCK_LPI2c5
Clock gate name: LPI2C5

enumerator kCLOCK_LPI2c6
Clock gate name: LPI2C6

enumerator kCLOCK_LPI2c7
Clock gate name: LPI2C7

enumerator kCLOCK_LPI2c8
Clock gate name: LPI2C8

enumerator kCLOCK_LPI2c9
Clock gate name: LPI2C9

enumerator kCLOCK_LPI2c10
Clock gate name: LPI2C10

enumerator kCLOCK_LPI2c11
Clock gate name: LPI2C11

enumerator kCLOCK_LPI2c12
Clock gate name: LPI2C12

enumerator kCLOCK_LPI2c13
Clock gate name: LPI2C13

enumerator kCLOCK_LPSpi2
Clock gate name: LPSPI2

enumerator kCLOCK_LPSpi3
Clock gate name: LPSPI3

enumerator kCLOCK_LPSpi4
Clock gate name: LPSPI4

enumerator kCLOCK_LPSpi5
Clock gate name: LPSPI5

enumerator kCLOCK_LPSpi6
Clock gate name: LPSPI6

enumerator kCLOCK_LPSpi7
Clock gate name: LPSPI7

enumerator kCLOCK_LPSpi8
Clock gate name: LPSPI8

enumerator kCLOCK_LPSpi9
Clock gate name: LPSPI9

enumerator kCLOCK_LPSpi10
Clock gate name: LPSPI10

enumerator kCLOCK_LPSpi11
Clock gate name: LPSPI11

enumerator kCLOCK_LPSpi12
Clock gate name: LPSPI12

enumerator kCLOCK_LPSpi13
Clock gate name: LPSPI13

enumerator kCLOCK_Sai0
Clock gate name: SAI0

enumerator kCLOCK_Sai1
Clock gate name: SAI1

enumerator kCLOCK_Sai2
Clock gate name: SAI2

enumerator kCLOCK_I3c0
Clock gate name: I3C0

enumerator kCLOCK_I3c1
Clock gate name: I3C1

enumerator kCLOCK_Crc0
Clock gate name: CRC0

enumerator kCLOCK_Wwdt0
Clock gate name: WWDT0

enumerator kCLOCK_Wwdt1
Clock gate name: WWDT1

enumerator kCLOCK_Ct32b0
Clock gate name: CTIMER0

enumerator kCLOCK_Ct32b1
Clock gate name: CTIMER1

enumerator kCLOCK_Ct32b2
Clock gate name: CTIMER2

enumerator kCLOCK_Ct32b3
Clock gate name: CTIMER3

enumerator kCLOCK_Ct32b4
Clock gate name: CTIMER4

enumerator kCLOCK_Mrt0
Clock gate name: Mrt0

enumerator kCLOCK_Utick0
Clock gate name: Utick0

enumerator kCLOCK_Sema424
Clock gate name: SEMA42_4

enumerator kCLOCK_Mu4
Clock gate name: MU4

enumerator kCLOCK_Pint0
Clock gate name: PINT0

enumerator kCLOCK_Pvts0
Clock gate name: PVTS0

enumerator kCLOCK_InputMux0
Clock gate name: PMUX_CMPT_SPLITTER

enumerator kCLOCK_Freqme0
Clock gate name: FREQME0

enumerator kCLOCK_SafoSgi
Clock gate name: SAFO_SGI

enumerator kCLOCK_Trace
Clock gate name: TRACE

enumerator kCLOCK_Prince0
Clock gate name: PRINCE0

enumerator kCLOCK_Prince1
Clock gate name: PRINCE1

enumerator kCLOCK_PrinceExe
Clock gate name: PRINCE_EXE

enumerator kCLOCK_Syspm0
Clock gate name: CMX_PERFMON0

enumerator kCLOCK_Syspm1
Clock gate name: CMX_PERFMON1

enumerator kCLOCK_Hifi4
Clock gate name: HIFI4

enumerator kCLOCK_Npu0
Clock gate name: NPU0

enumerator kCLOCK_CompAccessRamArbiter1
Clock gate name: COMP_ACCESS_RAM_ARBITER1

enumerator kCLOCK_Iopctl0
Clock gate name: IOMUXC_VDD2

enumerator kCLOCK_Hifi4AccessRamArbiter1
Clock gate name: HIFI4_ACCESS_RAM_ARBITER1

enumerator kCLOCK_MediaAccessRamArbiter0
Clock gate name: MEDIA_ACCESS_RAM_ARBITER0

enumerator kCLOCK_Sleepcon1
Clock gate name: SLEEPCONCPU1

enumerator kCLOCK_Syscon1
Clock gate name: SYSCONSENSE1

enumerator kCLOCK_SenseAccessRamArbiter0
Clock gate name: SENSE_ACCESS_RAM_ARBITER0

enumerator kCLOCK_Hifi1
Clock gate name: HIFI1

enumerator kCLOCK_Dma2
Clock gate name: DMA2

enumerator kCLOCK_Dma3
Clock gate name: DMA3

enumerator kCLOCK_LPFlexComm17
Clock gate name: LP_Flexcomm17

enumerator kCLOCK_LPFlexComm18
Clock gate name: LP_Flexcomm18

enumerator kCLOCK_LPFlexComm19
Clock gate name: LP_Flexcomm19

enumerator kCLOCK_LPFlexComm20
Clock gate name: LP_Flexcomm20

enumerator kCLOCK_LPI2c17
Clock gate name: LPI2C17

enumerator kCLOCK_LPI2c18
Clock gate name: LPI2C18

enumerator kCLOCK_LPI2c19
Clock gate name: LPI2C19

enumerator kCLOCK_LPI2c20
Clock gate name: LPI2C20

enumerator kCLOCK_LPSpi17
Clock gate name: LPSPI17

enumerator kCLOCK_LPSpi18
Clock gate name: LPSPI18

enumerator kCLOCK_LPSpi19
Clock gate name: LPSPI19

enumerator kCLOCK_LPSpi20
Clock gate name: LPSPI20

enumerator kCLOCK_LPUart17
Clock gate name: LPUART17

enumerator kCLOCK_LPUart18
Clock gate name: LPUART18

enumerator kCLOCK_LPUart19
Clock gate name: LPUART19

enumerator kCLOCK_LPUart20
Clock gate name: LPUART20

enumerator kCLOCK_Sai3
Clock gate name: SAI3

enumerator kCLOCK_I3c2
Clock gate name: I3C2

enumerator kCLOCK_I3c3
Clock gate name: I3C3

enumerator kCLOCK_Gpio8
Clock gate name: GPIO8

enumerator kCLOCK_Gpio9
Clock gate name: GPIO9

enumerator kCLOCK_Gpio10
Clock gate name: GPIO10

enumerator kCLOCK_Pint1
Clock gate name: PINT1

enumerator kCLOCK_Ct32b5
Clock gate name: CTIMER5

enumerator kCLOCK_Ct32b6
Clock gate name: CTIMER6

enumerator kCLOCK_Ct32b7
Clock gate name: CTIMER7

enumerator kCLOCK_Mrt1
Clock gate name: Mrt1

enumerator kCLOCK_Utick1
Clock gate name: Utick1

enumerator kCLOCK_Cdog3
Clock gate name: CDOG3

enumerator kCLOCK_Cdog4
Clock gate name: CDOG4

enumerator kCLOCK_Mu3
Clock gate name: MU3

enumerator kCLOCK_Sema423
Clock gate name: SEMA42_3

enumerator kCLOCK_Wwdt2
Clock gate name: WWDT2

enumerator kCLOCK_Wwdt3
Clock gate name: WWDT3

enumerator kCLOCK_Pvts1
Clock gate name: PINT1

enumerator kCLOCK_InputMux1
Clock gate name: PMUX_SNS_SPLITTER

enumerator kCLOCK_Syscon2
Clock gate name: SYSCON_COMM

enumerator kCLOCK_Iopctl2
Clock gate name: IOMUXC_VDDN

enumerator kCLOCK_Cpu1
Clock gate name: CPU1

enumerator kCLOCK_Mu0
Clock gate name: MU0

enumerator kCLOCK_Mu1
Clock gate name: MU1

enumerator kCLOCK_Mu2
Clock gate name: MU2

enumerator kCLOCK_OsTimer
Clock gate name: OsTimer

enumerator kCLOCK_Sema420
Clock gate name: SEMA42_0

enumerator kCLOCK_Sdadc0
Clock gate name: SDADC0

enumerator kCLOCK_Adc0
Clock gate name: SARADC0

enumerator kCLOCK_Acmp0
Clock gate name: Acmp0

enumerator kCLOCK_Pdm
Clock gate name: MICFIL(PDM)

enumerator kCLOCK_Glikey4
Clock gate name: GLIKEY_SYSCON1

enumerator kCLOCK_Dbg
Clock gate name: DBG

enumerator kCLOCK_Syscon3
Clock gate name: SYSCON_SENSE0

enumerator kCLOCK_Iopctl1
Clock gate name: IOMUXC_VDD1

enumerator kCLOCK_Glikey1
Clock gate name: GLIKEY1

enumerator kCLOCK_LPI2c15
Clock gate name: LPI2C

enumerator kCLOCK_MediaAccessRamArbiter1
Clock gate name: MEDIA_ACCESS_RAM_ARBITER1

enumerator kCLOCK_Axi0
Clock gate name: AXI0

enumerator kCLOCK_Gpu
Clock gate name: VGPU

enumerator kCLOCK_MipiDsiCtrl
Clock gate name: MIPIDSI

enumerator kCLOCK_LPSpi16
Clock gate name: LPSPI16

enumerator kCLOCK_LPSpi14
Clock gate name: LPSPI14

enumerator kCLOCK_Xspi2
Clock gate name: XSPI2

enumerator kCLOCK_Mmu2
Clock gate name: MMU2

enumerator kCLOCK_Glikey5
Clock gate name: GLIKEY_SYSCON2

enumerator kCLOCK_Flexio
Clock gate name: FLEXIO0

enumerator kCLOCK_Lcdif
Clock gate name: LCDIF(DCN)

enumerator kCLOCK_Syscon4
Clock gate name: SYSCONMEDIA

enumerator kCLOCK_JpgDecoder
Clock gate name: JPG_DECODER

enumerator kCLOCK_PngDecoder
Clock gate name: PNG_DECODER

enumerator kCLOCK_Ezhv
Clock gate name: EZHV

enumerator kCLOCK_AxbsEzh
Clock gate name: AXBS_EZH

enumerator kCLOCK_Glikey2
Clock gate name: GLIKEY2

enumerator kCLOCK_Usb0
Clock gate name: USB0

enumerator kCLOCK_Usb1
Clock gate name: USB1

enumerator kCLOCK_Usdhc0
Clock gate name: USHDC0

enumerator kCLOCK_Usdhc1
Clock gate name: USDHC1

enumerator kCLOCK_Gdet3Ref
Clock gate name: GDET3 Reference clock

enumerator kCLOCK_Rtc
RTC functional clock gating.

enumerator kCLOCK_Gdet3
GDET3 functional clock gating.

enumerator kCLOCK_UsbphyRef
USBPHY referrence clock gating.

enum _clock_name
Clock name used to get clock frequency.

Values:

enumerator kCLOCK_CoreSysClk
Core clock (aka system clock)

enumerator kCLOCK_BusClk
Bus clock (AHB/APB clock, aka HCLK)

enumerator kCLOCK_MclkClk
MCLK, to MCLK pin

enumerator kCLOCK_Wdt2Clk
Watchdog1

enumerator kCLOCK_Wdt3Clk
Watchdog1

enumerator kCLOCK_Hifi1CpuClk
Hifi1 DSP clock

enumerator kCLOCK_LPFlexComm17Clk
Flexcomm17Clock

enumerator kCLOCK_LPFlexComm18Clk
Flexcomm18Clock

enumerator kCLOCK_LPFlexComm19Clk
Flexcomm19Clock

enumerator kCLOCK_LPFlexComm20Clk
Flexcomm20Clock

enumerator kCLOCK_Vdd1ClockOutClk
VDD1 CLOCKOUT

enumerator kCLOCK_AdcClk
ADC

enumerator kCLOCK_Xspi2Clk
XSPI1

enumerator kCLOCK_SctClk
SCT

enumerator kCLOCK_SystickClk
Systick

enumerator kCLOCK_Sdio0Clk
SDIO0

enumerator kCLOCK_Sdio1Clk
SDIO1

enumerator kCLOCK_I3cClk
I3C

enumerator kCLOCK_Usb0Clk
USB0

enumerator kCLOCK_Usb1Clk
USB1

enumerator kCLOCK_PdmClk
Digital Mic clock

enumerator kCLOCK_AcmpClk
Acmp clock

enumerator kCLOCK_FlexioClk
FlexIO

enumerator kCLOCK_LPSpi14Clk
LPSPI14

enumerator kCLOCK_LPI2c15Clk
LPI2C15

enumerator kCLOCK_LPSpi16Clk
LPSPI16

enumerator kCLOCK_VgpuClk
VGPU Core

enumerator kCLOCK_LcdifClk
LCDIF Clock

enumerator kCLOCK_MipiDphyClk
MIPI D-PHY Bit Clock

enumerator kCLOCK_MipiDphyEscRxClk
MIPI D-PHY RX Clock

enumerator kCLOCK_MipiDphyEscTxClk
MIPI D-PHY TX Clock

enum __clock_attach_id
The enumerator of clock attach Id for multi sources clock slices. Not all Muxer has MUX output enable control, if no mux output enable control, the mux output will be always enabled.

Values:

enumerator kFRO1_DIV3_to_SENSE_BASE
Attach FRO1 divided-by-3 clock to Sense Base Clock.

enumerator kFRO1_DIV1_to_SENSE_BASE
Attach FRO1 max clock to Sense Base Clock.

enumerator kFRO2_DIV3_to_SENSE_BASE
Attach FRO2 divided-by-3 clock to Sense Base Clock.

enumerator kLPOSC_to_SENSE_BASE
Attach LPOSC to Sense Base Clock.

enumerator kSENSE_BASE_to_SENSE_DSP
Attach Sense base clock to Sense DSP Clock.

enumerator kFRO2_DIV1_to_SENSE_DSP
Attach FRO2 max clock to Sense DSP Clock.

enumerator kAUDIO_PLL_PFD1_to_SENSE_DSP
Attach Audio PLL PFD1 clock to Sense DSP Clock.

enumerator kFRO1_DIV1_to_SENSE_DSP
Attach FRO1 max clock to Sense DSP Clock.

enumerator kNONE_to_SENSE_DSP
Attach NONE to Sense DSP Clock.

enumerator kSENSE_BASE_to_SAI3
Attach Sense base clock to SAI3 Clock.

enumerator kFRO2_DIV8_to_SAI3
Attach FRO2_DIV8 clock to SAI3 Clock.

enumerator kFRO2_DIV1_to_SAI3
Attach FRO2 clock to SAI3 Clock.

enumerator kAUDIO_VDD1_to_SAI3
Attach audio_clk to SAI3 Clock.

enumerator kNONE_to_SAI3
Attach NONE to SAI3 Clock.

enumerator kSENSE_BASE_to_UTICK
Attach Sense base clock to UTICK Functional Clock.

enumerator kLPOSC_to_UTICK
Attach LPOSC clock to UTICK Functional Clock.

enumerator kFRO2_DIV1_to_UTICK
Attach FRO2 max clock to UTICK Functional Clock.

enumerator kFRO1_DIV2_to_UTICK
Attach FRO1_DIV2 clock to UTICK Functional Clock.

enumerator kNONE_to_UTICK
Attach NONE to UTICK Functional Clock.

enumerator kLPOSC_to_WWDT2
Attach LPOSC clock to WWDT2 Functional Clock.

enumerator kNONE_to_WWDT2
Attach NONE to WWDT2 Functional Clock.

enumerator kLPOSC_to_WWDT3
Attach LPOSC clock to WWDT3 Functional Clock.

enumerator kNONE_to_WWDT3
Attach NONE to WWDT3 Functional Clock.

enumerator kSENSE_BASE_to_SYSTICK
Attach Sense base clock to SYSTICK Functional Clock.

enumerator kLPOSC_to_SYSTICK
Attach LPOSC clock to SYSTICK Functional Clock.

enumerator k32KHZ_WAKE_to_SYSTICK
Attach 32 kHz wake clock(switch to other clock source before sense vdd1 enter SRPG mode.) to SYSTICK Functional Clock.

enumerator kOSC_CLK_to_SYSTICK
Attach OSC clock to SYSTICK Functional Clock.

enumerator kNONE_to_SYSTICK
Attach NONE to SYSTICK Functional Clock.

enumerator kSENSE_BASE_to_CTIMER5
Attach Sense base clock to CTIMER5 Clock.

enumerator kAUDIO_VDD1_to_CTIMER5
Attach audio_clk to CTIMER5 Clock.

enumerator kFRO2_DIV1_to_CTIMER5
Attach FRO2 Max clock to CTIMER5 Clock.

enumerator k32KHZ_WAKE_to_CTIMER5
Attach 32KHz wake clock to CTIMER5 Clock.

enumerator kNONE_to_CTIMER5
Attach NONE to CTIMER5 Clock.

enumerator kSENSE_BASE_to_CTIMER6
Attach Sense base clock to CTIMER6 Clock.

enumerator kAUDIO_VDD1_to_CTIMER6
Attach audio_clk to CTIMER6 Clock.

enumerator kFRO2_DIV1_to_CTIMER6
Attach FRO2 Max clock to CTIMER6 Clock.

enumerator k32KHZ_WAKE_to_CTIMER6
Attach 32KHz wake clock to CTIMER6 Clock.

enumerator kNONE_to_CTIMER6
Attach NONE to CTIMER6 Clock.

enumerator kSENSE_BASE_to_CTIMER7
Attach Sense base clock to CTIMER7 Clock.

enumerator kAUDIO_VDD1_to_CTIMER7
Attach audio_clk to CTIMER7 Clock.

enumerator kFRO2_DIV1_to_CTIMER7
Attach FRO2 Max clock to CTIMER7 Clock.

enumerator k32KHZ_WAKE_to_CTIMER7
Attach 32KHz wake clock to CTIMER7 Clock.

enumerator kNONE_to_CTIMER7
Attach NONE to CTIMER7 Clock.

enumerator kSENSE_BASE_to_I3C23
Attach Sense base clock to I3C2 and I3C3 Functional Clock.

enumerator kFRO2_DIV1_to_I3C23
Attach FRO2 max clock to I3C2 and I3C3 Functional Clock.

enumerator kFRO1_DIV8_to_I3C23
Attach FRO2_DIV8 clock to I3C2 and I3C3 Functional Clock.

enumerator kOSC_CLK_to_I3C23
Attach OSC clock to I3C2 and I3C3 Functional Clock.

enumerator kNONE_to_I3C23
Attach NONE to I3C2 and I3C3 Functional Clock.

enumerator kMCLK_to_AUDIO_VDD1
Attach MCLK IN from Pad to Audio VDD1 Clock.

enumerator kOSC_to_AUDIO_VDD1
Attach OSC clock to Audio VDD1 Clock.

enumerator kFRO2_DIV8_to_AUDIO_VDD1
Attach FRO2_DIV8 clock to Audio VDD1 Clock.

enumerator kAUDIO_PLL_PFD3_to_AUDIO_VDD1
Attach AUDIO PLL PFD3 clock to Audio VDD1 Clock.

enumerator kSENSE_BASE_to_FLEXCOMM17
Attach Sense base clock to FLEXCOMM17.

enumerator kFRO2_DIV1_to_FLEXCOMM17
Attach FRO2 max clock to FLEXCOMM17.

enumerator kFRO1_DIV1_to_FLEXCOMM17
Attach FRO1 max clock to FLEXCOMM17.

enumerator k32KHZ_WAKE_to_FLEXCOMM17
Attach 32k_wake_clk to FLEXCOMM17.

enumerator kNONE_to_FLEXCOMM17
Attach NONE to FLEXCOMM17.

enumerator kSENSE_BASE_to_FLEXCOMM18
Attach Sense base clock to FLEXCOMM18.

enumerator kFRO2_DIV1_to_FLEXCOMM18
Attach FRO2 max clock to FLEXCOMM18.

enumerator kFRO1_DIV1_to_FLEXCOMM18
Attach FRO1 max clock to FLEXCOMM18.

enumerator k32KHZ_WAKE_to_FLEXCOMM18
Attach 32k_wake_clk to FLEXCOMM18.

enumerator kNONE_to_FLEXCOMM18
Attach NONE to FLEXCOMM18.

enumerator kSENSE_BASE_to_FLEXCOMM19
Attach Sense base clock to FLEXCOMM19.

enumerator kFRO2_DIV1_to_FLEXCOMM19
Attach FRO2 max clock to FLEXCOMM19.

enumerator kFRO1_DIV1_to_FLEXCOMM19
Attach FRO1 max clock to FLEXCOMM19.

enumerator k32KHZ_WAKE_to_FLEXCOMM19
Attach 32k_wake_clk to FLEXCOMM19.

enumerator kNONE_to_FLEXCOMM19
Attach NONE to FLEXCOMM19.

enumerator kSENSE_BASE_to_FLEXCOMM20
Attach Sense base clock to FLEXCOMM20.

enumerator kFRO2_DIV1_to_FLEXCOMM20
Attach FRO2 max clock to FLEXCOMM20.

enumerator kFRO1_DIV1_to_FLEXCOMM20
Attach FRO1 max clock to FLEXCOMM20.

enumerator k32KHZ_WAKE_to_FLEXCOMM20
Attach 32k_wake_clk to FLEXCOMM20.

enumerator kNONE_to_FLEXCOMM20
Attach NONE to FLEXCOMM20.

enumerator kCOMMON_BASE_to_COMMON_VDDN
Attach Common base clock to Common VDDN Clock.

enumerator kMAIN_PLL_PFD3_to_COMMON_VDDN
Attach Main PLL PFD3 clock to Common VDDN Clock.

enumerator kFRO1_DIV1_to_COMMON_VDDN
Attach FRO1 max clock to Common VDDN Clock.

enumerator kOSC_CLK_to_COMMON_VDDN
Attach OSC clock to Common VDDN Clock.

enumerator kOSC_CLK_to_USB_24MHZ
Attach OSC_CLK clock to 24MHz USB OSC Clock.

enumerator kFRO1_DIV8_to_USB_24MHZ
Attach FRO1_DIV8 clock to 24MHz USB OSC Clock.

enumerator kFRO1_DIV3_to_COMMON_BASE
Attach Fro1 divided-by-3 to Common Base Clock.

enumerator kFRO1_DIV1_to_COMMON_BASE
Attach Fro1 divided-by-1 to Common Base Clock.

enumerator kFRO0_DIV3_to_COMMON_BASE
Attach Fro0 divided-by-3 to Common Base Clock.

enumerator kLPOSC_to_COMMON_BASE
Attach LPOSC to Common Base Clock.

enumerator kOSC_CLK_to_EUSB_24MHZ
Attach OSC_CLK clock to 24MHz eUSB OSC Clock.

enumerator kFRO1_DIV8_to_EUSB_24MHZ
Attach FRO1_DIV8 clock to 24MHz eUSB OSC Clock.

enumerator kFRO1_DIV8_to_MAIN_PLL0
Attach FRO1_DIV8 clock to Main PLL0 Clock.

enumerator kOSC_CLK_to_MAIN_PLL0
Attach OSC_CLK clock to Main PLL0 Clock.

enumerator kFRO1_DIV8_to_AUDIO_PLL0
Attach FRO1_DIV8 to Audio PLL0 Clock.

enumerator kOSC_CLK_to_AUDIO_PLL0
Attach OSC_CLK clock (User-Selectable) to Audio PLL0 Clock.

enumerator kSENSE_BASE_to_SENSE_MAIN
Attach Sense base clock to Sense main clock.

enumerator kFRO2_DIV1_to_SENSE_MAIN
Attach FRO2 max clock to Sense main clock.

enumerator kAUDIO_PLL_PFD3_to_SENSE_MAIN
Attach Audio PLL PFD3 clock to Sense main clock.

enumerator kFRO1_DIV1_to_SENSE_MAIN
Attach FRO1 max clock to Sense main clock.

enumerator kSENSE_BASE_to_SENSE_RAM
Attach Sense base clock to Sense RAM clock.

enumerator kFRO2_DIV1_to_SENSE_RAM
Attach FRO2 max clock to Sense RAM clock.

enumerator kAUDIO_PLL_PFD2_to_SENSE_RAM
Attach Audio PLL PFD2 clock to Sense RAM clock.

enumerator kFRO1_DIV1_to_SENSE_RAM
Attach FRO1 max clock to Sense RAM clock.

enumerator kSENSE_BASE_to_OSTIMER
Attach Sense Base to OSTIMER.

enumerator k32KHZ_WAKE_to_OSTIMER
Attach 32k_wake_clk to OSTIMER.

enumerator kLPOSC_to_OSTIMER
Attach LPOSC to OSTIMER.

enumerator kOSC_CLK_to_OSTIMER
Attach OSC clock to OSTIMER.

enumerator kNONE_to_OSTIMER
Attach NONE to OSTIMER.

enumerator kSENSE_BASE_to_SDADC
Attach Sense base clock to SDADC Functional Clock.

enumerator kFRO2_DIV8_to_SDADC
Attach FRO2_DIV8 clock to SDADC Functional Clock.

enumerator kAUDIO_PLL_to_SDADC
Attach Audio PLL VCO clock to SDADC Functional Clock.

enumerator kAUDIO_VDD1_to_SDADC
Attach audio_clk to SDADC Functional Clock.

enumerator kNONE_to_SDADC
Attach NONE to SDADC Functional Clock.

enumerator kSENSE_BASE_to_ADC
Attach Sense base clock to SARADC Functional Clock.

enumerator kMAIN_PLL_PFD1_to_ADC
Attach main_pll_pfd1 to SARADC Functional Clock.

enumerator kFRO2_DIV1_to_ADC
Attach FRO0 max clock to SARADC Functional Clock.

enumerator kOSC_CLK_to_ADC
Attach osc_clk to SARADC Functional Clock.

enumerator kNONE_to_ADC
Attach NONE to SARADC Functional Clock.

enumerator kOSC32K_to_32K_WAKE
Attach OSC 32K to Wake 32k Clock.

enumerator kLPOSC_DIV32_to_32K_WAKE
Attach LPOSC clock divided by 32 by default to Wake 32k Clock.

enumerator kNONE_to_32K_WAKE
Attach NONE to Wake 32k Clock.

enumerator kSENSE_BASE_to_MICFIL0
Attach Sense base clock to MICFIL0 Functional Clock.

enumerator kAUDIO_PLL_PFD3_to_MICFIL0
Attach Audio PLL PFD3 clock to MICFIL0 Functional Clock.

enumerator kFRO2_DIV1_to_MICFIL0
Attach FRO2 MAX clock to MICFIL0 Functional Clock.

enumerator kAUDIO_VDD1_to_MICFIL0
Attach Audio clock to MICFIL0 Functional Clock.

enumerator kNONE_to_MICFIL0
Attach NONE to MICFIL0 Functional Clock.

enumerator kSENSE_BASE_to_LPI2C15
Attach Sense base clock to LPI2C15 Functional Clock.

enumerator kFRO1_DIV1_to_LPI2C15
Attach FRO1 max clock to LPI2C15 Functional Clock.

enumerator kFRO1_DIV2_to_LPI2C15
Attach FRO1 divided-by-2 to LPI2C15 Functional Clock.

enumerator kFRO2_DIV1_to_LPI2C15
Attach FRO2 max clock to LPI2C15 Functional Clock.

enumerator kNONE_to_LPI2C15
Attach NONE to LPI2C15 Functional Clock.

enumerator kSENSE_BASE_to_VDD1_CLKOUT
Attach Sense VDD1 Base Clock to VDD1 CLKOUT Clock.

enumerator kAUDIO_PLL_PFD3_to_VDD1_CLKOUT
Attach AUDIO PLL PFD3 Clock to VDD1 CLKOUT Clock.

enumerator kFRO2_DIV1_to_VDD1_CLKOUT
Attach FRO2 Max Clock to VDD1 CLKOUT Clock.

enumerator kFRO1_DIV1_to_VDD1_CLKOUT
Attach FRO1 Max Clock to VDD1 CLKOUT Clock.

enumerator kNONE_to_VDD1_CLKOUT
Attach NONE to VDD1 CLKOUT Clock.

enumerator kMEDIA_VDDN_BASE_to_MEDIA_VDDN
Attach MEDIA VDDN base clock to MEDIA VDDN clock.

enumerator kMAIN_PLL_PFD0_to_MEDIA_VDDN
Attach MAIN PLL PFD0 to MEDIA VDDN clock.

enumerator kFRO0_DIV1_to_MEDIA_VDDN
Attach FRO0 max to MEDIA VDDN clock.

enumerator kMAIN_PLL_PFD2_to_MEDIA_VDDN
Attach MAIN PLL PFD2 to MEDIA VDDN clock.

enumerator kMEDIA_VDD2_BASE_to_MEDIA_MAIN
Attach MEDIA VDD2 base clock to MEDIA MAIN clock.

enumerator kMAIN_PLL_PFD0_to_MEDIA_MAIN
Attach Main PLL PFD0 clock to MEDIA MAIN clock.

enumerator kFRO0_DIV1_to_MEDIA_MAIN
Attach FRO0 to MEDIA MAIN clock.

enumerator kMAIN_PLL_PFD2_to_MEDIA_MAIN
Attach Main PLL PFD2 clock to MEDIA MAIN clock.

enumerator kFRO1_DIV3_to_MEDIA_VDDN_BASE
Attach Fro1 divided-by-3 to Media VDDN Base Clock.

enumerator kFRO1_DIV1_to_MEDIA_VDDN_BASE
Attach Fro1 divided-by-1 to Media VDDN Base Clock.

enumerator kFRO0_DIV3_to_MEDIA_VDDN_BASE
Attach Fro0 divided-by-3 to Media VDDN Base Clock.

enumerator kLPOSC_to_MEDIA_VDDN_BASE
Attach LPOSC clock to Media VDDN Base Clock.

enumerator kFRO1_DIV3_to_MEDIA_VDD2_BASE
Attach Fro1 divided-by-3 to Media VDD2 Base Clock.

enumerator kFRO1_DIV1_to_MEDIA_VDD2_BASE
Attach Fro1 divided-by-1 to Media VDD2 Base Clock.

enumerator kFRO0_DIV3_to_MEDIA_VDD2_BASE
Attach Fro0 divided-by-3 to Media VDD2 Base Clock.

enumerator kLPOSC_to_MEDIA_VDD2_BASE
Attach LPOSC clock to Media VDD2 Base Clock.

enumerator kCOMMON_BASE_to_XSPI2
Attach Common VDDN base clock to XSPI2 Functional Clock.

enumerator kAUDIO_PLL_PFD1_to_XSPI2
Attach Audio PLL PFD1 clock to XSPI2 Functional Clock.

enumerator kFRO0_DIV1_to_XSPI2
Attach FRO0 max clock to XSPI2 Functional Clock.

enumerator kMAIN_PLL_PFD3_to_XSPI2
Attach Main PLL PFD3 clock to XSPI2 Functional Clock.

enumerator kNONE_to_XSPI2
Attach NONE to XSPI2 Functional Clock.

enumerator k32KHZ_WAKE_to_USB
Attach Wakeup 32K to USB Functional Clock.

enumerator kLPOSC_to_USB
Attach LPOSC_1M to USB Functional Clock.

enumerator kUSB_24MHZ_to_USB
Attach 24MHz USB OSC clock to USB Functional Clock.

enumerator kNONE_to_USB
Attach NONE to USB Functional Clock.

enumerator k32KHZ_WAKE_to_EUSB
Attach Wakeup 32K to eUSB Functional Clock.

enumerator kLPOSC_to_EUSB
Attach LPOSC_1M to eUSB Functional Clock.

enumerator kEUSB_24MHZ_to_EUSB
Attach 24MHz eUSB OSC clock to eUSB Functional Clock.

enumerator kNONE_to_EUSB
Attach NONE to eUSB Functional Clock.

enumerator kMEDIA_VDDN_BASE_to_SDIO0
Attach Media VDDN base clock to SDIO0 Functional Clock.

enumerator kAUDIO_PLL_PFD0_to_SDIO0
Attach Audio PLL PFD0 clock to SDIO0 Functional Clock.

enumerator kFRO0_DIV1_to_SDIO0
Attach FRO0 max clock to SDIO0 Functional Clock.

enumerator kMAIN_PLL_PFD2_to_SDIO0
Attach Main PLL PFD2 clock to SDIO0 Functional Clock.

enumerator kNONE_to_SDIO0
Attach NONE to SDIO0 Functional Clock.

enumerator kMEDIA_VDDN_BASE_to_SDIO1
Attach Media VDDN base clock to SDIO1 Functional Clock.

enumerator kAUDIO_PLL_PFD0_to_SDIO1
Attach Audio PLL PFD0 clock to SDIO1 Functional Clock.

enumerator kFRO0_DIV1_to_SDIO1
Attach FRO0 max clock to SDIO1 Functional Clock.

enumerator kMAIN_PLL_PFD1_to_SDIO1
Attach Main PLL PFD1 clock to SDIO1 Functional Clock.

enumerator kNONE_to_SDIO1
Attach NONE to SDIO1 Functional Clock.

enumerator kMEDIA_VDD2_BASE_to_MIPI_DSI_HOST_PHY
Attach Media VDD2 base clock to MIPI_DSI_Host PHY Clock.

enumerator kFRO0_DIV1_to_MIPI_DSI_HOST_PHY
Attach FRO0 max clock to MIPI_DSI_Host PHY Clock.

enumerator kAUDIO_PLL_PFD2_to_MIPI_DSI_HOST_PHY
Attach Audio PLL PFD2 clock to MIPI_DSI_Host PHY Clock.

enumerator kNONE_to_MIPI_DSI_HOST_PHY
Attach NONE to MIPI_DSI_Host PHY Clock.

enumerator kMEDIA_VDD2_BASE_to_MIPI_DPHYESC_CLK
Attach Media VDD2 base clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kMAIN_PLL_PFD1_to_MIPI_DPHYESC_CLK
Attach Main PLL PFD1 clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kFRO0_DIV1_to_MIPI_DPHYESC_CLK
Attach FRO0 max clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kAUDIO_PLL_PFD2_to_MIPI_DPHYESC_CLK
Attach Audio PLL PFD2 clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kNONE_to_MIPI_DPHYESC_CLK
Attach NONE to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kMEDIA_VDD2_BASE_to_VGPU
Attach Media VDD2 base clock to VGPU Clock.

enumerator kMAIN_PLL_PFD0_to_VGPU
Attach Main PLL PFD0 clock to VGPU Clock.

enumerator kFRO0_DIV1_to_VGPU
Attach FRO0 max clock to VGPU Clock.

enumerator kMAIN_PLL_PFD2_to_VGPU
Attach Main PLL PFD2 clock to VGPU Clock.

enumerator kNONE_to_VGPU
Attach NONE to VGPU Clock.

enumerator kMEDIA_VDD2_BASE_to_LPSPI14
Attach Media VDD2 base clock to LPSPI14 Clock.

enumerator kFRO0_DIV1_to_LPSPI14
Attach FRO0 max clock to LPSPI14 Clock.

enumerator kMAIN_PLL_PFD0_to_LPSPI14
Attach Main PLL PFD0 clock to LPSPI14 Clock.

enumerator kFRO1_DIV1_to_LPSPI14
Attach FRO1 max clock to LPSPI14 Clock.

enumerator kNONE_to_LPSPI14
Attach NONE to LPSPI14 Clock.

enumerator kMEDIA_VDD2_BASE_to_LPSPI16
Attach Media VDD2 base clock to LPSPI16 Clock.

enumerator kFRO0_DIV1_to_LPSPI16
Attach FRO0 max clock to LPSPI16 Clock.

enumerator kMAIN_PLL_PFD0_to_LPSPI16
Attach Main PLL PFD0 clock to LPSPI16 Clock.

enumerator kFRO1_DIV1_to_LPSPI16
Attach FRO1 max clock to LPSPI16 Clock.

enumerator kNONE_to_LPSPI16
Attach NONE to LPSPI16 Clock.

enumerator kMEDIA_VDD2_BASE_to_FLEXIO
Attach MEDIA VDD2 base clock to FLEXIO.

enumerator kFRO0_DIV1_to_FLEXIO
Attach FRO0 max clock to FLEXIO.

enumerator kFRO1_DIV1_to_FLEXIO
Attach FRO1 max clock to FLEXIO.

enumerator kMAIN_PLL_PFD3_to_FLEXIO
Attach Main PLL PFD3 clock to FLEXIO.

enumerator kNONE_to_FLEXIO
Attach NONE to FLEXIO.

enumerator kMEDIA_VDD2_BASE_to_LCDIF
Attach MEDIA VDD2 base clock to LCDIF.

enumerator kMAIN_PLL_PFD2_to_LCDIF
Attach Main PLL PFD0 clock to LCDIF.

enumerator kFRO0_DIV1_to_LCDIF
Attach FRO0 max clock to LCDIF.

enumerator kAUDIO_PLL_PFD1_to_LCDIF
Attach Audio PLL PFD1 to LCDIF.

enumerator kNONE_to_LCDIF
Attach NONE to LCDIF.

enum _clock_div_name
Clock dividers.

Values:

enumerator kCLOCK_DivLPFlexComm17Clk
LP_FLEXCOMM17 Clk Divider.

enumerator kCLOCK_DivLPFlexComm18Clk
LP_FLEXCOMM18 Clk Divider.

enumerator kCLOCK_DivLPFlexComm19Clk
LP_FLEXCOMM19 Clk Divider.

enumerator kCLOCK_DivLPFlexComm20Clk
LP_FLEXCOMM20 Clk Divider.

enumerator kCLOCK_DivSenseDspClk
Sense DSP Clk Divider.

enumerator kCLOCK_DivSai3Clk
SAI3 FCLK Clk Divider.

enumerator kCLOCK_DivUtick1Clk
UTICK1 Functional Clk Divider.

enumerator kCLOCK_DivSystickClk
SYSTICK Functional Clk Divider.

enumerator kCLOCK_DivCtimer5Clk
CTIMER5 FCLK Clk Divider.

enumerator kCLOCK_DivCtimer6Clk
CTIMER6 FCLK Clk Divider.

enumerator kCLOCK_DivCtimer7Clk
CTIMER7 FCLK Clk Divider.

enumerator kCLOCK_DivI3c23Clk
I3C2 and I3C3 Functional Clk Divider.

enumerator kCLOCK_DivCommonVddnClk
Common VDDN Clk Divider.

enumerator kCLOCK_DivSenseMainClk
VDD1_SENSE Main Clock Divider.

enumerator kCLOCK_DivSenseRamClk
Senese RAM Clk Divider.

enumerator kCLOCK_DivOstimerClk
OSTIMER Clk Divider.

enumerator kCLOCK_DivSdadcClk
SDADC Clk Divider.

enumerator kCLOCK_DivAdcClk
SARADC Clk Divider.

enumerator kCLOCK_DivMicfil0Clk
DMIC0 Clk Divider.

enumerator kCLOCK_DivLpi2c15Clk
PMIC LPI2C Clk Divider.

enumerator kCLOCK_DivVdd1ClockOut
CLKOUT_VDD1 Clk Divider.

enumerator kCLOCK_DivMediaVddnClk
Media VDDN Clk Divider.

enumerator kCLOCK_DivMediaMainClk
Media Main Clk Divider.

enumerator kCLOCK_DivXspi2Clk
XSPI2 Clk Divider.

enumerator kCLOCK_DivSdio0Clk
SDIO0 Clk Divider.

enumerator kCLOCK_DivSdio1Clk
SDIO1 Clk Divider.

enumerator kCLOCK_DivDphyClk
DPHY Clk Divider.

enumerator kCLOCK_DivDphyEscRxClk
DphyEscRx Clk Divider.

enumerator kCLOCK_DivDphyEscTxClk
DphyEscTx Clk Divider.

enumerator kCLOCK_DivVgpuClk
VGPU Clk Divider.

enumerator kCLOCK_DivLpspi14Clk
LPSPI14 Clk Divider.

enumerator kCLOCK_DivLpspi16Clk
LPSPI16 Clk Divider.

enumerator kCLOCK_DivFlexioClk
FLEXIO Clk Divider.

enumerator kCLOCK_DivLcdifClk
LCDIF Pixel Clk Divider.

enumerator kCLOCK_DivLowFreqClk
Low frequency Clk Divider.

enum __clock_fro_output_en
FRO output enable.

Values:

enumerator kCLOCK_FroDiv1OutEn
Enable Fro Div1 output.

enumerator kCLOCK_FroDiv2OutEn
Enable Fro Div2 output.

enumerator kCLOCK_FroDiv3OutEn
Enable Fro Div3 output.

enumerator kCLOCK_FroDiv6OutEn
Enable Fro Div6 output.

enumerator kCLOCK_FroDiv8OutEn
Enable Fro Div8 output.

enumerator kCLOCK_FroAllOutEn
Enable all Fro output.

enum _clock_fro_interrupt
FRO Interrupt control.

Values:

enumerator kCLOCK_FroTrimUpdateReqInt
Trim Update Request Interrupt Enable.

enumerator kCLOCK_FroTuneErrInt
Tune Error Interrupt Enable.

enumerator kCLOCK_FroLossOfLockInt
Loss-of-Lock Error Interrupt Enable.

enumerator kCLOCK_FroAllIntterrupt
All Interrupt Enable.

enum _clock_fro_flag
FRO flags.

Values:

enumerator kCLOCK_FroTuneOnceDone
FRO Tune Once Done Flag.

enumerator kCLOCK_FroTrimLock
FRO Trim Lock Flag.

enumerator kCLOCK_FroTrimUpdateReq
FRO Trim Update Request Flag.

enumerator kCLOCK_FroTuneErr
FRO Tune Error Flag.

enumerator kCLOCK_FroLossOfLockErr
FRO Loss-of-lock Error Flag.

enum _clock_domain_enable
Clock Control for each power domain.

Values:

enumerator kCLOCK_Vdd2CompDomainEnable
Clock Control of VDD2_COMP Domain.

enumerator kCLOCK_Vdd1SenseDomainEnable
Clock Control of VDD1_SENSE Domain.

enumerator kCLOCK_Vdd2DspDomainEnable
Clock Control of VDD2_DSP Domain.

enumerator kCLOCK_Vdd2MediaDomainEnable
Clock Control of VDD2_MEDIA Domain.

enumerator kCLOCK_VddnMediaDomainEnable
Clock Control of VDDN_MEDIA Domain.

enumerator kCLOCK_Vdd2ComDomainEnable
Clock Control of VDD2_COM Domain.

enumerator kCLOCK_VddnComDomainEnable
Clock Control of VDDN_COM Domain.

enumerator kCLOCK_AllDomainEnable
Clock Control of all Domain.

enum _clock_pfd
PLL PFD clock name.

Values:

enumerator kCLOCK_Pfd0
PLL PFD0

enumerator kCLOCK_Pfd1
PLL PFD1

enumerator kCLOCK_Pfd2
PLL PFD2

enumerator kCLOCK_Pfd3
PLL PFD3

enum _main_pll_src
MainPLL Reference Input Clock Source.

Values:

enumerator kCLOCK_MainPllFro1Div8
FRO1_DIV8 clock

enumerator kCLOCK_MainPllOscClk
OSC clock

enum _main_pll_mult
MainPLL Multiplication Factor.

Values:

enumerator kCLOCK_MainPllMult16
Divide by 16

enumerator kCLOCK_MainPllMult17
Divide by 17

enumerator kCLOCK_MainPllMult18
Divide by 18

enumerator kCLOCK_MainPllMult19
Divide by 19

enumerator kCLOCK_MainPllMult20
Divide by 20

enumerator kCLOCK_MainPllMult21
Divide by 21

enumerator kCLOCK_MainPllMult22
Divide by 22

enum _audio_pll_src
AudioPll Reference Input Clock Source.

Values:

enumerator kCLOCK_AudioPllFro1Div8
FRO1_DIV8 clock

enumerator kCLOCK_AudioPllOscClk
OSC clock

enum _audio_pll_mult
AudioPll Multiplication Factor.

Values:

enumerator kCLOCK_AudioPllMult16
Divide by 16

enumerator kCLOCK_AudioPllMult17
Divide by 17

enumerator kCLOCK_AudioPllMult18
Divide by 18

enumerator kCLOCK_AudioPllMult19
Divide by 19

enumerator kCLOCK_AudioPllMult20
Divide by 20

enumerator kCLOCK_AudioPllMult21
Divide by 21

enumerator kCLOCK_AudioPllMult22
Divide by 22

enum _osc32k_cap_trim
Capacitor Trim Value for OSC32KNP.

Values:

enumerator kCLOCK_Osc32kCapPf0
Capacitor Trim Value 0pF.

enumerator kCLOCK_Osc32kCapPf2
Capacitor Trim Value 2pF.

enumerator kCLOCK_Osc32kCapPf4
Capacitor Trim Value 4pF.

enumerator kCLOCK_Osc32kCapPf6
Capacitor Trim Value 6pF.

enumerator kCLOCK_Osc32kCapPf8
Capacitor Trim Value 8pF.

enumerator kCLOCK_Osc32kCapPf10
Capacitor Trim Value 10pF.

enumerator kCLOCK_Osc32kCapPf12
Capacitor Trim Value 12pF.

enumerator kCLOCK_Osc32kCapPf14
Capacitor Trim Value 14pF.

enumerator kCLOCK_Osc32kCapPf16
Capacitor Trim Value 16pF.

enumerator kCLOCK_Osc32kCapPf18
Capacitor Trim Value 18pF.

enumerator kCLOCK_Osc32kCapPf20
Capacitor Trim Value 20pF.

enumerator kCLOCK_Osc32kCapPf22
Capacitor Trim Value 22pF.

enumerator kCLOCK_Osc32kCapPf24
Capacitor Trim Value 24pF.

enumerator kCLOCK_Osc32kCapPf26
Capacitor Trim Value 26pF.

enumerator kCLOCK_Osc32kCapPf28
Capacitor Trim Value 28pF.

enumerator kCLOCK_Osc32kCapPf30
Capacitor Trim Value 30pF.

typedef enum *_clock_usb_src* clock_usb_src_t
USB clock source definition.

typedef enum *_clock_usb_phy_src* clock_usb_phy_src_t
Source of the USB HS PHY.

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

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

typedef enum *_clock_attach_id* clock_attach_id_t
The enumerator of clock attach Id for multi sources clock slices. Not all Muxer has MUX output enable control, if no mux output enable control, the mux output will be always enabled.

typedef enum *_clock_div_name* clock_div_name_t
Clock dividers.

typedef enum *_clock_fro_output_en* clock_fro_output_en_t
FRO output enable.

typedef struct *_clock_fro_config* clock_fro_config_t

FRO configuration.

typedef enum *_clock_domain_enable* clock_domain_enable_t

Clock Control for each power domain.

typedef enum *_clock_pfd* clock_pfd_t

PLL PFD clock name.

typedef enum *_main_pll_src* main_pll_src_t

MainPLL Reference Input Clock Source.

typedef enum *_main_pll_mult* main_pll_mult_t

MainPLL Multiplication Factor.

typedef struct *_clock_main_pll_config* clock_main_pll_config_t

PLL configuration for MAINPLL.

typedef enum *_audio_pll_src* audio_pll_src_t

AudioPll Reference Input Clock Source.

typedef enum *_audio_pll_mult* audio_pll_mult_t

AudioPll Multiplication Factor.

typedef struct *_clock_audio_pll_config* clock_audio_pll_config_t

PLL configuration for Audio PLL.

typedef enum *_osc32k_cap_trim* osc32k_cap_trim_t

Capacitor Trim Value for OSC32KNP.

typedef struct *_clock_osc32k_config_t* clock_osc32k_config_t

configuration for 32K OSC

volatile uint32_t g_xtalFreq

External XTAL (SYSOSC) clock frequency.

The XTAL (SYSOSC) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetXtalFreq to set the value in to clock driver. For example, if XTAL is 16MHz,

```
CLOCK_SetXtalFreq(160000000);
```

volatile uint32_t g_clkinFreq

External CLK_IN pin clock frequency (clkin).

The CLK_IN pin (clkin) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetClkinFreq to set the value in to clock driver. For example, if CLK_IN is 16MHz,

```
CLOCK_SetClkinFreq(160000000);
```

volatile uint32_t g_32kClkinFreq

External 32KHz input clock frequency.

The External 32KHz input clock frequency in Hz, used when OSC32KNP in bypass mode, use the function CLOCK_Set32kClkinFreq to set the value in to clock driver. For example, if EXTALIN32K input is 32768,

```
CLOCK_Set32kClkinFreq(32768);
```

volatile uint32_t g_mclkFreq

External MCLK IN clock frequency.

The MCLK IN clock frequency in Hz, when the clock is setup, use the function CLOCK_SetMclkFreq to set the value in to clock driver. For example, if MCLK IN is 16MHz,

```
CLOCK_SetMclkFreq(160000000);
```

void CLOCK_EnableClock(*clock_ip_name_t* clk)

Enable the clock for specific IP.

Parameters

- clk – : Clock to be enabled.

Returns

Nothing

void CLOCK_DisableClock(*clock_ip_name_t* clk)

Disable the clock for specific IP.

Parameters

- clk – : Clock to be disabled.

Returns

Nothing

void CLOCK_AttachClk(*clock_attach_id_t* connection)

brief Configure the clock selection muxes.

For some of the muxes, there's SEL_EN bit to gate the mux ouput to reduce power, using kNONT_to_XXX to gate the mux oupput. used by any peripheral.

Parameters

- connection – : Clock to be configured.

Returns

Nothing

void 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. Divided clock frequency = Undivided clock frequency / divider.

Returns

Nothing

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Return Frequency of selected clock.

Returns

Frequency of selected clock

uint32_t CLOCK_GetSenseAudioClkFreq(void)

Return Frequency of VDD1 audio clk.

Returns

Frequency of audio_clk

uint32_t CLOCK_GetHifi1ClkFreq(void)

Return Frequency of Sense DSP clk.

Returns

Frequency of Sense DSP clk

void CLOCK_EnableFroClkOutput(FRO_Type *base, uint32_t divOutEnable)

Enable/Disable FRO clock output.

Parameters

- base – : base address of FRO.
- divOutEnable – : Or'ed value of clock_fro_output_en_t to enable certain clock freq output.

void CLOCK_ConfigFroTrim(FRO_Type *base, uint16_t trimVal)

Configure FRO trim values when FRO is configured in Open loop mode.

Parameters

- base – : base address of FRO.
- trimVal – : 12bits trim value.

void CLOCK_DisableFro(FRO_Type *base)

Disable the FRO clock. This API will disable the FRO clock output and power off FRO.

Parameters

- base – : base address of FRO.

status_t CLOCK_EnableFroAutoTuning(FRO_Type *base, const *clock_fro_config_t* *config, bool enable)

Enable/Disable FRO for close loop mode(autotuner).

Parameters

- base – : base address of FRO.
- config – : The configuration for FRO.
- enable – Enable auto tuning mode or not.

Return values

- kStatus_Success – successfully tuned to the target configuration.
- kStatus_InvalidArgument – Invalid arguement.
- kStatus_Fail – failed to lock to the target frequency.

void CLOCK_EnableFroClkFreq(FRO_Type *base, uint32_t targetFreq, uint32_t divOutEnable)

Enable FRO clock output with specified frequency.

Parameters

- base – : base address of FRO.
- targetFreq – target fro frequency.
- divOutEnable – Or'ed value of clock_fro_output_en_t to enable certain clock freq output.

status_t CLOCK_EnableFroClkFreqCloseLoop(FRO_Type *base, const *clock_fro_config_t* *config, uint32_t divOutEnable)

Enable FRO clock output with target frequency using FRO close loop mode. For example, to enable FRO2 to ouput 200MHZ:

```
const clock_fro_config_t config = {
 .targetFreq = 200000000U,
 .range = 100U,
 .trim1DelayUs = 15U,
```

(continues on next page)

(continued from previous page)

```
    .trim2DelayUs = 150U,
    .refDiv = 0U,
    .enableInt = 0U,
    .coarseTrimEn = true,
};
CLOCK_EnableFroClkFreqCloseLoop(FRO2, &config, kCLOCK_FroAllOutEn);
```

If the API returns kStatus_Timeout or kStatus_Fail, the CLOCK_FroTuneOnce and trim value from AUTOTRIM register can be used to get the desired frequency in open loop mode.

Parameters

- base – : base address of FRO.
- config – : The configuration for FRO.
- divOutEnable – : Or'ed value of clock_fro_output_en_t to enable certain clock freq output.

Return values

- kStatus_Success – successfully tuned to the target configuration.
- kStatus_InvalidArgument – Invalid arguement.
- kStatus_Timeout – Timeout to lock to the target frequency. The config->range may too strict(Suggested range >=100) or the auto tuner can't lock to the desired frequency.
- kStatus_Fail – The FRO locked with error.

uint32_t CLOCK_FroTuneOnce(FRO_Type *base, uint16_t trimVal)

FRO Tune once. Use FRO tuneonce feature to calculate the FRO output frequency with given trimVal. The FRO should be powered up and reference clock(SOSC) should be enabled.

Parameters

- base – : base address of FRO.
- trimVal – : trim value.

Returns

FRO frequency in Hz.

void CLOCK_FroFineTune(FRO_Type *base, uint32_t targetFreq, uint16_t trimVal)

Fine tune FRO output frequency in open loop mode. The FRO should be powered up and reference clock(SOSC) should be enabled. The API uses the FRO tuneonce feature to try trim values from (trimVal-1, trimVal, trimVal+1) and select a best one.

Parameters

- base – : base address of FRO.
- targetFreq – : Target frequency in Hz.
- trimVal – : trim value. When CLOCK_EnableFroClkFreqCloseLoop failed to lock, the value in FRO AUTOTRIM register can be used to do fine tune.

uint32_t CLOCK_GetFroFlags(FRO_Type *base)

Get FRO flags.

Parameters

- base – : base address of FRO.
- flags – Or'ed value of _clock_fro_flag.

static inline void CLOCK_ClearFroFlags(FRO_Type *base, uint32_t flags)

Clear FRO flags.

Parameters

- base – : base address of FRO.
- flags – Or'ed value of _clock_fro_flag to clear.

static inline void CLOCK_EnableFroFastStartup(FRO_Type *base, bool enable)

Enable/Disable FRO fast startup.

Parameters

- base – : base address of FRO.
- true (enable,) – to enable fast startup, false to disable fast startup.

uint32_t CLOCK_GetFroClkFreq(uint32_t id)

Return Frequency of FRO clk.

Parameters

- id – FRO index

Returns

Frequency of FRO clk

uint32_t CLOCK_GetSenseMainClkFreq(void)

Return Frequency of sense main clk.

Returns

Frequency of main clk

uint32_t CLOCK_GetMainPllFreq(void)

Return Frequency of MAINPLL.

Returns

Frequency of MAINPLL

uint32_t CLOCK_GetMainPfdFreq(*clock_pfd_t* pfd)

Get current output frequency of specific Main PLL PFD.

Parameters

- pfd – : pfd name to get frequency.

Returns

Frequency of MainPLL PFD.

uint32_t CLOCK_GetAudioPllFreq(void)

Return Frequency of AUDIO PLL.

Returns

Frequency of AUDIO PLL

uint32_t CLOCK_GetAudioPfdFreq(*clock_pfd_t* pfd)

Get current output frequency of specific Audio PLL PFD.

Parameters

- pfd – : pfd name to get frequency.

Returns

Frequency of AUDIO PLL PFD.

static inline uint32_t CLOCK_GetXtalInClkFreq(void)

Return Frequency of High-Freq output of FRO.

Return Frequency of sys osc Clock

Returns
Frequency of High-Freq output of FRO

Returns
Frequency of sys osc Clock. Or CLK_IN pin frequency.

uint32_t CLOCK_GetSenseBaseClkFreq(void)

Return Frequency of Sense Base clock.

Returns
Frequency of sense base clk

void CLOCK_EnableSysOscClk(bool enable, bool enableLowPower, uint32_t delay_us)

Enable/Disable sys osc clock from external crystal clock.

Parameters

- enable – : true to enable system osc clock, false to bypass system osc.
- enableLowPower – : true to enable low power mode, false to enable high gain mode.
- delay_us – : Delay time after OSC power up.

uint32_t CLOCK_GetVddnComBaseClkFreq(void)

Return Frequency of VDDN_COM Base Clock.

Returns
Frequency of baseclk_comn

uint32_t CLOCK_GetVddnMediaBaseClkFreq(void)

Return Frequency of VDDN_MEDIA Base Clock.

Returns
Frequency of baseclk_mdn

uint32_t CLOCK_GetVdd2MediaBaseClkFreq(void)

Return Frequency of VDD2_MEDIA Base Clock.

Returns
Frequency of baseclk_md2

uint32_t CLOCK_GetMediaMainClkFreq(void)

Return Frequency of MEDIA_MAIN Clock.

Returns
Frequency of media_main_clk

uint32_t CLOCK_GetMediaVddnClkFreq(void)

Return Frequency of MEDIA_VDDN Clock.

Returns
Frequency of media_vddn_clk

void CLOCK_EnableFro2ClkForDomain(uint32_t domainEnable)

Enable/Disable FRO2 clock for various domains.

Parameters

- domainEnable – : Or'ed value of clock_domain_enable_t to enable DRFO2 max clock for certain domain.

void CLOCK_EnableMainPllPfdClkForDomain(*clock_pfd_t* pfd, uint32_t domainEnable)

Enable/Disable MainPLL PFD clock for various domains.

Enables PFD clock of MainPLL for various domains. Each PFD clock of MainPLL can be sent to various domain. To reduce power consumption, turn off the PFD in the domain when this PFD is not configured.

Parameters

- pfd – : Which PFD clock to control.
- domainEnable – : Or'ed value of clock_domain_enable_t to enable clock for certain domain.

void CLOCK_EnableAudioPllPfdClkForDomain(*clock_pfd_t* pfd, uint32_t domainEnable)

Enable/Disable AUDIO PFD clock for various domains.

Enables PFD clock of AUDIO for various domains. Each PFD clock of AUDIO can be sent to various domain. To reduce power consumption, turn off the PFD in the domain when this PFD is not configured.

Parameters

- pfd – : Which PFD clock to control.
- domainEnable – : Or'ed value of clock_domain_enable_t to enable clock for certain domain.

void CLOCK_EnableAudioPllVcoClkForDomain(uint32_t domainEnable)

Enable/Disable AUDIO VCO clock for various domains.

VCO clock of Audio PLL can be sent to various domains to reduce power consumption when VCO is not set to use in such domain. This VCO can be disabled for such domain.

Parameters

- domainEnable – : Or'ed value of clock_domain_enable_t to enable clock for certain domain.

void CLOCK_InitMainPll(const *clock_main_pll_config_t* *config)

Initialize the Main PLL.

Parameters

- config – : Configuration to set to PLL.

static inline void CLOCK_DeinitMainPll(void)

Deinit the Main PLL.

Parameters

- none. –

status_t CLOCK_InitMainPfd(*clock_pfd_t* pfd, uint8_t divider)

Initialize the Main PLL PFD.

Note: It is recommended that PFD settings are kept between 12-35.

Parameters

- pfd – : Which PFD clock to enable.
- divider – : The PFD divider value.

Returns

kStatus_Success if successfully, kStatus_Timeout if timeout happen.

static inline void CLOCK_DeinitMainPfd(*clock_pfd_t* pfd)

Disable the Main PLL PFD.

Parameters

- pfd – : Which PFD clock to disable.

void CLOCK_InitAudioPll(const *clock_audio_pll_config_t* *config)

Initialize the audio PLL.

Parameters

- config – : Configuration to set to PLL.

static inline void CLOCK_DeinitAudioPll(void)

Deinit the Audio PLL.

Parameters

- none. –

status_t CLOCK_InitAudioPfd(*clock_pfd_t* pfd, uint8_t divider)

Initialize the audio PLL PFD.

Note: It is recommended that PFD settings are kept between 12-35.

Parameters

- pfd – : Which PFD clock to enable.
- divider – : The PFD divider value.

Returns

kStatus_Success if successfully, kStatus_Timeout if timeout happen.

static inline void CLOCK_DeinitAudioPfd(*clock_pfd_t* pfd)

Disable the audio PLL PFD.

Parameters

- pfd – : Which PFD clock to disable.

static inline uint32_t CLOCK_GetLpOscFreq(void)

Return Frequency of Lower power osc.

Returns

Frequency of LPOSC

static inline uint32_t CLOCK_GetSysOscFreq(void)

Return Frequency of sys osc Clock.

Returns

Frequency of sys osc Clock. Or CLK_IN pin frequency.

static inline uint32_t CLOCK_GetMclkInClkFreq(void)

Return Frequency of MCLK Input Clock.

Returns

Frequency of MCLK input Clock.

uint32_t CLOCK_GetOsc32KFreq(void)

Return Frequency of 32kHz osc.

Returns

Frequency of 32kHz osc

void CLOCK_EnableOsc32K(*clock_osc32k_config_t* *config)

Enables OSC32KNP.

Parameters

- config – : configuration set to OSC32KNP

static inline void CLOCK_DisableOsc32K(void)

Disable OSC32KNP.

uint32_t CLOCK_GetWakeClk32KFreq(void)

Return Frequency of 32khz wake clk.

Returns

Frequency of 32kHz wake clk

static inline void CLOCK_SetXtalFreq(uint32_t freq)

Set the XTALIN (system OSC) frequency based on board setting. NOTE, when SOSC is used, either CLOCK_SetXtalFreq or CLOCK_SetClkinFreq(But NOT both.) should be called to tell driver the clock frequency connected to SOSC module.

Parameters

- freq – : The XTAL input clock frequency in Hz.

static inline void CLOCK_SetClkinFreq(uint32_t freq)

Set the CLKIN (CLKIN pin) frequency based on board setting.

Parameters

- freq – : The CLK_IN pin input clock frequency in Hz.

static inline void CLOCK_Set32kClkinFreq(uint32_t freq)

Set the 32KHz external input frequency based on board setting.

Parameters

- freq – : The 32KHz external pin input clock frequency in Hz.

static inline void CLOCK_SetMclkFreq(uint32_t freq)

Set the MCLK IN frequency based on board setting.

Parameters

- freq – : The MCLK input clock frequency in Hz.

static inline uint32_t CLOCK_GetCoreSysClkFreq(void)

Return Frequency of Core/system clock.

Returns

Frequency of core or system Clock

uint32_t CLOCK_GetXspiClkFreq(uint32_t id)

Return Frequency of XSPI function clock.

Parameters

- id – : XSPI index to get frequency.

Returns

Frequency of XSPI functional Clock

uint32_t CLOCK_GetUtickClkFreq(void)

Return Frequency of UTICK function clock.

Returns

Frequency of UTICK functional Clock

uint32_t CLOCK_GetSystickClkFreq(void)

Return Frequency of systick clk.

Returns
Frequency of systick clk

uint32_t CLOCK_GetWdtClkFreq(uint32_t id)

Return Frequency of WDT clk.

Parameters

- id – : WDT index to get frequency.

Returns
Frequency of WDT clk

uint32_t CLOCK_GetAcmpClkFreq(void)

Return Frequency of ACMP clk.

Returns
Frequency of ACMP clk

uint32_t CLOCK_GetSaiClkFreq(void)

Return Frequency of SAI clk.

Returns
Frequency of SAI clk

uint32_t CLOCK_GetUsbClkFreq(void)

Return Frequency of USB clk.

Returns
Frequency of USB clk

uint32_t CLOCK_GetI3cClkFreq(void)

Return Frequency of I3C clk.

Returns
Frequency of I3C clk

uint32_t CLOCK_GetLPFlexCommClkFreq(uint32_t id)

Return Frequency of Flexcomm functional Clock.

Parameters

- id – : flexcomm index to get frequency.

Returns
Frequency of Flexcomm functional Clock

uint32_t CLOCK_GetLPI2cClkFreq(uint32_t id)

Return Frequency of LPI2C functional Clock.

Parameters

- id – : LPI2C index to get frequency.

Returns
Frequency of LPI2C functional Clock

uint32_t CLOCK_GetLPSpiClkFreq(uint32_t id)

Return Frequency of LPSPI functional Clock.

Parameters

- id – : LPSPI index to get frequency.

Returns
Frequency of LPSPI functional Clock

uint32_t CLOCK_GetFlexioClkFreq(void)

Return Frequency of Flexio functional Clock.

Returns
Frequency of Flexcomm functional Clock

uint32_t CLOCK_GetOSTimerClkFreq(void)

Return Frequency of OSTIMER functional Clock.

Returns
Frequency of OSTIMER functional Clock

uint32_t CLOCK_GetMicfilClkFreq(void)

Return Frequency of MICFIL functional Clock.

Returns
Frequency of MICFIL functional Clock

uint32_t CLOCK_GetCTimerClkFreq(uint32_t id)

Return Frequency of Ctimer Clock.

Parameters

- id – : ctimer index to get frequency.

Returns
Frequency of Ctimer Clock

uint32_t CLOCK_GetVdd1ClockOutClkFreq(void)

Return Frequency of VDD1 Clock Out.

Returns
Frequency of ClockOut of sense domain

uint32_t CLOCK_GetAdcClkFreq(void)

Return Frequency of Adc Clock.

Returns
Frequency of Adc Clock.

uint32_t CLOCK_GetSdAdcClkFreq(void)

Return Frequency of SDADC Clock.

Returns
Frequency of SDADC Clock.

uint32_t CLOCK_GetVgpuClkFreq(void)

Return Frequency of VGPU functional Clock.

Returns
Frequency of VGPU functional Clock

uint32_t CLOCK_GetLcdifClkFreq(void)

Return Frequency of LCDIF pixel Clock.

Returns
Frequency of LCDIF pixel Clock

uint32_t CLOCK_GetMipiDphyClkFreq(void)

Return Frequency of MIPI DPHY functional Clock.

Returns
Frequency of MIPI DPHY functional Clock

uint32_t CLOCK_GetMipiDphyEscRxClkFreq(void)

Return Frequency of MIPI DPHY Esc RX functional Clock.

Returns

Frequency of MIPI DPHY Esc RX functional Clock

uint32_t CLOCK_GetMipiDphyEscTxClkFreq(void)

Return Frequency of MIPI DPHY Esc Tx functional Clock.

Returns

Frequency of MIPI DPHY Esc Tx functional Clock

uint32_t CLOCK_GetUsdhcClkFreq(uint32_t id)

Return Frequency of USDHC Clock.

Parameters

- id – : uSDHC index to get frequency.

Returns

Frequency of USDHC Clock

bool CLOCK_EnableUsbhs0PhyPllClock(*clock_usb_phy_src_t* src, uint32_t freq)

Enable USB HS PHY PLL clock.

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

Parameters

- src – USB HS PHY PLL clock source.
- freq – The frequency specified by src.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

bool CLOCK_EnableUsbhs0Clock(*clock_usb_src_t* src, uint32_t freq)

Enable USB HS clock.

This function only enables the access to USB HS prepheral, upper layer should first call the ref CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

- src – USB HS does not care about the clock source, here must be ref kCLOCK_UsbSrcUnused.
- freq – USB HS does not care about the clock source, so this parameter is ignored.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

void CLOCK_DisableUsbhs0PhyPllClock(void)

Disable USB HS PHY PLL clock.

This function disables USB HS PHY PLL clock.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.5.1.

CLOCK_DeinitSysPfd

CLOCK_InitSysPfd

CDOG_CLOCKS
: Clock ip name array for CDOG.

FREQME_CLOCKS
: Clock ip name array for FREQME.

GPU_CLOCKS
: Clock ip name array for GPU.

MIPI_DSI_HOST_CLOCKS
: Clock ip name array for MIPI DSI.

LCDIF_CLOCKS
: Clock ip name array for LCDIF.

SCT_CLOCKS
: Clock ip name array for SCT.

USB_CLOCKS
: Clock ip name array for USB.

USBPHY_CLOCKS
: Clock ip name array for USBPHY.

XSPI_CLOCKS
: Clock ip name array for FlexSPI.

MMU_CLOCKS
: Clock ip name array for MMU.

CACHE64_CLOCKS
: Clock ip name array for CACHE64.

CACHE64POLSEL_CLOCKS
: Clock ip name array for CACHE64_POLSEL.

LPADC_CLOCKS
: Clock ip name array for ADC.

SDADC_CLOCKS
: Clock ip name array for SDADC.

CMP_CLOCKS
: Clock ip name array for ACMP.

USDHC_CLOCKS
: Clock ip name array for uSDHC.

WWDT_CLOCKS
: Clock ip name array for WWDT.

UTICK_CLOCKS
: Clock ip name array for UTICK.

FLEXIO_CLOCKS
: Clock ip name array for FlexIO.

LP_FLEXCOMM_CLOCKS
: Clock ip name array for LP_FLEXCOMM.

LPUART_CLOCKS
: Clock ip name array for LPUART.

LPI2C_CLOCKS
: Clock ip name array for LPI2C.

LPSPI_CLOCKS
: Clock ip name array for LSPI.

SAI_CLOCKS
: Clock ip name array for SAI.

SEMA42_CLOCKS
: Clock ip name array for SEMA.

CRC_CLOCKS
: Clock ip name array for MU.

  Clock ip name array for CRC.

GPIO_CLOCKS
: Clock ip name array for GPIO.

PDM_CLOCKS
: Clock ip name array for PDM.

PINT_CLOCKS
: Clock ip name array for PINT.

PNGDEC_CLOCKS
: Clock ip name array for PNGDEC.

JPEGDEC_CLOCKS
: Clock ip name array for JPEGDEC.

I3C_CLOCKS
: Clock ip name array for I3C.

MRT_CLOCKS
: Clock ip name array for MRT.

CTIMER_CLOCKS
: Clock ip name array for CT32B.

OSTIMER_CLOCKS
: Clock ip name array for OSTIMER.

RTC_CLOCKS
: Clock ip name array for RTC.

CLK_GATE_REG_OFFSET_SHIFT
: Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

CLK_GATE_REG_OFFSET_MASK

CLK_GATE_BIT_SHIFT_SHIFT

CLK_GATE_BIT_SHIFT_MASK

CLK_GATE_DEFINE(reg_offset, bit_shift)

CLK_GATE_ABSTRACT_REG_OFFSET(x)

CLK_GATE_ABSTRACT_BITS_SHIFT(x)

CLK_CTL0_PSCCTL0

CLK_CTL0_PSCCTL1

CLK_CTL0_PSCCTL2

CLK_CTL0_PSCCTL3

CLK_CTL0_PSCCTL4

CLK_CTL0_PSCCTL5

CLK_CTL1_PSCCTL0

CLK_CTL1_PSCCTL1

CLK_CTL2_PSCCTL0

CLK_CTL3_PSCCTL0

CLK_CTL4_PSCCTL0

CLK_CTL4_PSCCTL1

SYSCON3_SEC_CLK_CTRL

CLKCTL0_ONE_SRC_CLKSLICE_ENABLE

CLKCTL3_ONE_SRC_CLKSLICE_ENABLE

CLKCTL4_ONE_SRC_CLKSLICE_ENABLE

CLK_MUX_INST_INDEX_SHIFT

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

[31-29 14 12:11 10:0] [CLKCTL index]: [Disable MUX output]: [MUXA choice]:[MUXA offset]. MUX offset 0 means end of descriptor.

CLK_MUX_CHOICE_OFFSET_SHIFT

CLK_MUX_DISABLE_OUTPUT_MASK

SENSEBASECLKSEL_OFFSET

SENSEDSPCPUCLKDIV_OFFSET

SENSEDSPCPUCLKSEL_OFFSET

SAI3FCLKSEL_OFFSET

SAI3CLKDIV_OFFSET

UTICK1FCLKSEL_OFFSET

UTICK1CLKDIV_OFFSET

WWDT2FCLKSEL_OFFSET

WWDT3FCLKSEL_OFFSET

SYSTICKFCLKSEL_OFFSET

SYSTICKFCLKDIV_OFFSET

CTIMER5FCLKSEL_OFFSET

CTIMER6FCLKSEL_OFFSET

CTIMER7FCLKSEL_OFFSET

CTIMER5CLKDIV_OFFSET

CTIMER6CLKDIV_OFFSET

CTIMER7CLKDIV_OFFSET

I3C23FCLKSEL_OFFSET

I3C23FCLKDIV_OFFSET

FC17FCLKSEL_OFFSET

FC18FCLKSEL_OFFSET

FC19FCLKSEL_OFFSET

FC20FCLKSEL_OFFSET

FC17FCLKDIV_OFFSET

FC18FCLKDIV_OFFSET

FC19FCLKDIV_OFFSET

FC20FCLKDIV_OFFSET

AUDIOVDD1CLKSEL_OFFSET

USBCLKSRC24MCLKSEL_OFFSET

USBCLKSRC24MCLKSEL_OFFSET

COMMONVDDNCLKSEL_OFFSET

COMMONVDDNCLKDIV_OFFSET

COMNBASECLKSEL_OFFSET

EUSBCLKSRC24MCLKSEL_OFFSET

MAINPLL0CLKSEL_OFFSET

MAINPLL0LOCKTIMEDIV2_OFFSET

AUDIOPLL0CLKSEL_OFFSET

AUDIOPLL0LOCKTIMEDIV2_OFFSET

SENSEMAINCLKDIV_OFFSET

SENSE_MAINCLKSEL_OFFSET

SENSERAMCLKSEL_OFFSET

SENSERAMCLKDIV_OFFSET

OSEVENTTFCLKSEL_OFFSET

OSEVENTFCLKDIV_OFFSET

SDADCFCLKSEL_OFFSET

SDADCFCLKDIV_OFFSET

SARADCFCLKSEL_OFFSET

SARADCFCLKDIV_OFFSET

WAKE32KCLKSEL_OFFSET

MICFIL0FCLKSEL_OFFSET

MICFIL0FCLKDIV_OFFSET

LPI2CFCLKSEL_OFFSET

LPI2CFCLKDIV_OFFSET

SENSE_CLKOUTCLKSEL_OFFSET

SENSE_CLKOUTCLKDIV_OFFSET

MEDIAVDDNCLKSEL_OFFSET

MEDIAVDDNCLKDIV_OFFSET

MEDIAMAINCLKSEL_OFFSET

MEDIAMAINCLKDIV_OFFSET

MDNBASECLKSEL_OFFSET

MD2BASECLKSEL_OFFSET

XSPI2FCLKSEL_OFFSET

XSPI2FCLKDIV_OFFSET

USBFCLKSEL_OFFSET

EUSBFCLKSEL_OFFSET

SDIO0FCLKSEL_OFFSET

SDIO0FCLKDIV_OFFSET

SDIO1FCLKSEL_OFFSET

SDIO1FCLKDIV_OFFSET

DPHYCLKSEL_OFFSET

DPHYCLKDIV_OFFSET

DPHYESCCLKSEL_OFFSET

DPHYESCRXCLKDIV_OFFSET

DPHYESCTXCLKDIV_OFFSET

VGPUCLKSEL_OFFSET

VGPUCLKDIV_OFFSET

LPSPI14CLKSEL_OFFSET

LPSPI14CLKDIV_OFFSET

LPSPI16CLKSEL_OFFSET

LPSPI16CLKDIV_OFFSET

FLEXIOCLKSEL_OFFSET

FLEXIOCLKDIV_OFFSET

LCDIFPIXELCLKSEL_OFFSET

LCDIFPIXELCLKDIV_OFFSET

LOWFREQCLKDIV_OFFSET

CLKCTL0_INDEX

CLKCTL1_INDEX

CLKCTL2_INDEX

CLKCTL3_INDEX

CLKCTL4_INDEX

CLKCTL0_TUPLE_MUXA(reg, choice)

CLKCTL1_TUPLE_MUXA(reg, choice)

CLKCTL2_TUPLE_MUXA(reg, choice)

CLKCTL3_TUPLE_MUXA(reg, choice)

CLKCTL4_TUPLE_MUXA(reg, choice)

CLKCTL0_TUPLE_MUXA_NONE(reg, choice)
Macro for gated clock mux

CLKCTL1_TUPLE_MUXA_NONE(reg, choice)

CLKCTL2_TUPLE_MUXA_NONE(reg, choice)

CLKCTL3_TUPLE_MUXA_NONE(reg, choice)

CLKCTL4_TUPLE_MUXA_NONE(reg, choice)

CLKCTL_TUPLE_REG(base, tuple)

CLKCTL_TUPLE_SEL(tuple)

uint32_t targetFreq
Target frequency.

uint16_t refDiv
OSC Reference clock divider. 1 for divide ratio 1.

uint16_t trim1DelayUs
Trim 1 delay in us, minimum is 15us. Used when running in Closed Loop mode and trim values are updated by 1 decimal unit.

uint16_t trim2DelayUs

Trim 2 delay in us. Used at start of closed loop mode when auto tuner is updating trim values by 16 decimal unit steps

uint8_t range

Trim Expected Count Range. Specifies the + or - counts that the FRO frequency can be off from TEXPCNT to be considerred locked. The value/100 is the % deviation.

uint32_t enableInt

Enable interrupts. Bit mask of _clock_fro_interrupt.

bool coarseTrimEn

Coarse Trim Enable. Set to true to allow autotrimming of the FRO high-byte trim bits.

bool fastStartupEn

Fast startup Enable. Enables FRO fast startup.

main_pll_src_t main_pll_src

Reference Input Clock Source.

uint32_t numerator

30 bit numerator of fractional loop divider.

uint32_t denominator

30 bit numerator of fractional loop divider.

main_pll_mult_t main_pll_mult

Multiplication Factor.

audio_pll_src_t audio_pll_src

Reference Input Clock Source.

uint32_t numerator

30 bit numerator of fractional loop divider.

uint32_t denominator

30 bit numerator of fractional loop divider.

audio_pll_mult_t audio_pll_mult

Multiplication Factor.

bool enableVcoOut

Enable VCO output

bool bypass

Bypass enable.

bool monitorEnable

Clock Monitor Enable.

bool lowPowerMode

Low-Power (Nano-Power) mode enable. NOTE, can only change from High-Power mode to Nano-Power mode, and not vice versa.

osc32k_cap_trim_t cap

Capacitor Trim Value.

struct _clock_fro_config

#include <fsl_clock.h> FRO configuration.

struct _clock_main_pll_config

#include <fsl_clock.h> PLL configuration for MAINPLL.

struct _clock_audio_pll_config

#include <fsl_clock.h> PLL configuration for Audio PLL.

struct _clock_osc32k_config_t

#include <fsl_clock.h> configuration for 32K OSC

2.5 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- Version 2.0.4
  - Release peripheral from reset if necessary in init function.
- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues
- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

enum _crc_bits

CRC bit width.

Values:

enumerator kCrcBits16

Generate 16-bit CRC code

enumerator kCrcBits32

Generate 32-bit CRC code

enum _crc_result

CRC result type.

Values:

enumerator kCrcFinalChecksum

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

enumerator kCrcIntermediateChecksum

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

typedef enum *_crc_bits* crc_bits_t

CRC bit width.

typedef enum *_crc_result* crc_result_t

CRC result type.

typedef struct *_crc_config* crc_config_t

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

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

Enables and configures the CRC peripheral module.

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

Parameters

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

static inline void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

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

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

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

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

Parameters

- config – CRC protocol configuration structure.

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

Writes data to the CRC module.

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

Parameters

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

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

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

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

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

Parameters

- base – CRC peripheral address.

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

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

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

2.6 CTIMER: Standard counter/timers

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

Ungates the clock and configures the peripheral for basic operation.

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

Parameters

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

void CTIMER_Deinit(CTIMER_Type *base)

Gates the timer clock.

Parameters

- base – Ctimer peripheral base address

void CTIMER_GetDefaultConfig(*ctimer_config_t* *config)

Fills in the timers configuration structure with the default settings.

The default values are:

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

Parameters

- config – Pointer to the user configuration structure.

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

Configures the PWM signal parameters.

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

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

---

Parameters

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

Returns

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

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

Configures the PWM signal parameters.

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

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

Parameters

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

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

Updates the pulse period of an active PWM signal.

Parameters

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

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

Updates the duty cycle of an active PWM signal.

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

Parameters

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

Returns

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

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

Enables the selected Timer interrupts.

Parameters

- base – Ctimer peripheral base address

- mask – The interrupts to enable. This is a logical OR of members of the enumeration ctimer_interrupt_enable_t

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

Disables the selected Timer interrupts.

Parameters

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

static inline uint32_t CTIMER_GetEnabledInterrupts(CTIMER_Type *base)

Gets the enabled Timer interrupts.

Parameters

- base – Ctimer peripheral base address

Returns

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

static inline uint32_t CTIMER_GetStatusFlags(CTIMER_Type *base)

Gets the Timer status flags.

Parameters

- base – Ctimer peripheral base address

Returns

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

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

Clears the Timer status flags.

Parameters

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

static inline void CTIMER_StartTimer(CTIMER_Type *base)

Starts the Timer counter.

Parameters

- base – Ctimer peripheral base address

static inline void CTIMER_StopTimer(CTIMER_Type *base)

Stops the Timer counter.

Parameters

- base – Ctimer peripheral base address

FSL_CTIMER_DRIVER_VERSION

Version 2.3.3

enum _ctimer_capture_channel

List of Timer capture channels.

Values:

enumerator kCTIMER_Capture_0
Timer capture channel 0

enumerator kCTIMER_Capture_1
Timer capture channel 1

enumerator kCTIMER_Capture_3
Timer capture channel 3

enum _ctimer_capture_edge
List of capture edge options.

Values:

enumerator kCTIMER_Capture_RiseEdge
Capture on rising edge

enumerator kCTIMER_Capture_FallEdge
Capture on falling edge

enumerator kCTIMER_Capture_BothEdge
Capture on rising and falling edge

enum _ctimer_match
List of Timer match registers.

Values:

enumerator kCTIMER_Match_0
Timer match register 0

enumerator kCTIMER_Match_1
Timer match register 1

enumerator kCTIMER_Match_2
Timer match register 2

enumerator kCTIMER_Match_3
Timer match register 3

enum _ctimer_external_match
List of external match.

Values:

enumerator kCTIMER_External_Match_0
External match 0

enumerator kCTIMER_External_Match_1
External match 1

enumerator kCTIMER_External_Match_2
External match 2

enumerator kCTIMER_External_Match_3
External match 3

enum _ctimer_match_output_control
List of output control options.

Values:

enumerator kCTIMER_Output_NoAction
No action is taken

enumerator kCTIMER_Output_Clear
Clear the EM bit/output to 0

enumerator kCTIMER_Output_Set
Set the EM bit/output to 1

enumerator kCTIMER_Output_Toggle
Toggle the EM bit/output

enum _ctimer_timer_mode
List of Timer modes.

Values:

enumerator kCTIMER_TimerMode

enumerator kCTIMER_IncreaseOnRiseEdge

enumerator kCTIMER_IncreaseOnFallEdge

enumerator kCTIMER_IncreaseOnBothEdge

enum _ctimer_interrupt_enable
List of Timer interrupts.

Values:

enumerator kCTIMER_Match0InterruptEnable
Match 0 interrupt

enumerator kCTIMER_Match1InterruptEnable
Match 1 interrupt

enumerator kCTIMER_Match2InterruptEnable
Match 2 interrupt

enumerator kCTIMER_Match3InterruptEnable
Match 3 interrupt

enum _ctimer_status_flags
List of Timer flags.

Values:

enumerator kCTIMER_Match0Flag
Match 0 interrupt flag

enumerator kCTIMER_Match1Flag
Match 1 interrupt flag

enumerator kCTIMER_Match2Flag
Match 2 interrupt flag

enumerator kCTIMER_Match3Flag
Match 3 interrupt flag

enum ctimer_callback_type_t
Callback type when registering for a callback. When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Values:

enumerator kCTIMER_SingleCallback
Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently

enumerator kCTIMER_MultipleCallback

Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

typedef enum *_ctimer_capture_channel* ctimer_capture_channel_t

List of Timer capture channels.

typedef enum *_ctimer_capture_edge* ctimer_capture_edge_t

List of capture edge options.

typedef enum *_ctimer_match* ctimer_match_t

List of Timer match registers.

typedef enum *_ctimer_external_match* ctimer_external_match_t

List of external match.

typedef enum *_ctimer_match_output_control* ctimer_match_output_control_t

List of output control options.

typedef enum *_ctimer_timer_mode* ctimer_timer_mode_t

List of Timer modes.

typedef enum *_ctimer_interrupt_enable* ctimer_interrupt_enable_t

List of Timer interrupts.

typedef enum *_ctimer_status_flags* ctimer_status_flags_t

List of Timer flags.

typedef void (*ctimer_callback_t)(uint32_t flags)

typedef struct *_ctimer_match_config* ctimer_match_config_t

Match configuration.

This structure holds the configuration settings for each match register.

typedef struct *_ctimer_config* ctimer_config_t

Timer configuration structure.

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

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

void CTIMER_SetupMatch(CTIMER_Type *base, *ctimer_match_t* matchChannel, const *ctimer_match_config_t* *config)

Setup the match register.

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

- base – Ctimer peripheral base address
- matchChannel – Match register to configure
- config – Pointer to the match configuration structure

uint32_t CTIMER_GetOutputMatchStatus(CTIMER_Type *base, uint32_t matchChannel)

Get the status of output match.

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

Parameters

- base – Ctimer peripheral base address
- matchChannel – External match channel, user can obtain the status of multiple match channels at the same time by using the logic of "|" enumeration ctimer_external_match_t

Returns

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

void CTIMER_SetupCapture(CTIMER_Type *base, *ctimer_capture_channel_t* capture, *ctimer_capture_edge_t* edge, bool enableInt)

Setup the capture.

Parameters

- base – Ctimer peripheral base address
- capture – Capture channel to configure
- edge – Edge on the channel that will trigger a capture
- enableInt – Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

static inline uint32_t CTIMER_GetTimerCountValue(CTIMER_Type *base)

Get the timer count value from TC register.

Parameters

- base – Ctimer peripheral base address.

Returns

return the timer count value.

void CTIMER_RegisterCallBack(CTIMER_Type *base, *ctimer_callback_t* *cb_func, *ctimer_callback_type_t* cb_type)

Register callback.

This function configures CTimer Callback in following modes:

- Single Callback: cb_func should be pointer to callback function pointer For example: ctimer_callback_t ctimer_callback = pwm_match_callback; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback, kCTIMER_SingleCallback);
- Multiple Callback: cb_func should be pointer to array of callback function pointers Each element corresponds to Interrupt Flag in IR register. For example: ctimer_callback_t ctimer_callback_table[] = { ctimer_match0_callback, NULL, NULL, ctimer_match3_callback, NULL, NULL, NULL, NULL}; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback_table[0], kCTIMER_MultipleCallback);

Parameters

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

static inline void CTIMER_Reset(CTIMER_Type *base)

Reset the counter.

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

Parameters

- base – Ctimer peripheral base address

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

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

Parameters

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

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

Get capture channel value.

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

Parameters

- base – Ctimer peripheral base address
- capture – Select capture channel

Returns

The timer count capture value.

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

Enable reset match channel.

Set the specified match channel reset operation.

Parameters

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

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

Enable stop match channel.

Set the specified match channel stop operation.

Parameters

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

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

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

Parameters

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

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

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

Parameters

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

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

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

Parameters

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

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

Set the specified match shadow channel.

Parameters

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

struct __ctimer_match_config

#include <fsl_ctimer.h> Match configuration.

This structure holds the configuration settings for each match register.

Public Members

uint32_t matchValue

This is stored in the match register

bool enableCounterReset

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

bool enableCounterStop

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

ctimer_match_output_control_t outControl

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

bool outPinInitState

Initial value of the EM bit/output

bool enableInterrupt

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

struct _ctimer_config

#include <fsl_ctimer.h> Timer configuration structure.

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

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

Public Members

ctimer_timer_mode_t mode

Timer mode

ctimer_capture_channel_t input

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

uint32_t prescale

Prescale value

2.7 Dsp

enum _dsp_static_vec_sel

Fusion DSP vector table select.

Values:

enumerator kDSP_StatVecSel0

enumerator kDSP_StatVecSel1

typedef struct *_dsp_copy_image* dsp_copy_image_t

Structure for DSP copy image to destination address.

Defines start and destination address for copying image with given size.

typedef enum *_dsp_static_vec_sel* dsp_static_vec_sel_t

Fusion DSP vector table select.

void DSP_Init(void)

Initializing DSP core.

Power up DSP TCM Enable DSP clock Reset DSP peripheral

void DSP_Deinit(void)

Deinit DSP core.

Power down DSP TCM Disable DSP clock Set DSP peripheral reset

void DSP_CopyImage(*dsp_copy_image_t* *dspCopyImage)

Copy DSP image to destination address.

Copy DSP image from source address to destination address with given size.

Parameters

- dspCopyImage – Structure contains information for DSP copy image to destination address.

static inline void DSP_Start(void)

Start DSP core.

static inline void DSP_Stop(void)

Stop DSP core.

static inline void DSP_SetVecRemap(*dsp_static_vec_sel_t* statVecSel, uint32_t remap)

Set DSP static vector table remap.

Parameters

- statVecSel – static vector base address selection
- remap – static vector remap, For HiFi1 when statVecSel = 0, the valid range is 0~0x3FF, when statVecSel=1, the valid range is 0~0x7FF. For HiFi4, the bits value will remap ARADDR[22:10] when [STATVECSELECT] is high and ARADDR hits 1 KB from 2010_0000h.

FSL_DSP_DRIVER_VERSION

DSP driver version 2.0.0.

uint32_t *srcAddr

uint32_t *destAddr

uint32_t size

struct _dsp_copy_image

#include <fsl_dsp.h> Structure for DSP copy image to destination address.

Defines start and destination address for copying image with given size.

2.8 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;
EDMA_PrepareTransferTCD(handle, tcd, ....)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_PrepareTransferTCD(handle, tcd, ....)
```

(continues on next page)

(continued from previous page)

```
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

d. submit loop transfer

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

Parameters

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

Return values

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

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

Submits the eDMA transfer request.

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

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.10.7.

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull

TCD queue is full.

enumerator kStatus_EDMA_Busy

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

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes
Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes
Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes
Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes
Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes
Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA_TransferSize128Bytes
Source/Destination data transfer size is 128 bytes every time

enum _edma_modulo
eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable
Disable modulo

enumerator kEDMA_Modulo2bytes
Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes
Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes
Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes
Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes
Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone
No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle
eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type
Channel link type.

Values:

enumerator kEDMA_LinkNone
No channel link

enumerator kEDMA_MinorLink
Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.

Values:

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

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

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

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

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

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

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

_edma_interrupt_enable eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type
eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

enum edma_channel_memory_attribute
eDMA channel memory attribute

Values:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

enum _edma_channel_swap_size
eDMA4 channel swap size

Values:

enumerator kEDMA_ChannelSwapDisabled
Swap is disabled.

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

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

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

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

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

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

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

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

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

enum _edma_channel_access_type
eDMA4 channel access type

Values:

enumerator kEDMA_ChannelDataAccess
Data access for eDMA4 transfers.

enumerator kEDMA_ChannelInstructionAccess
Instruction access for eDMA4 transfers.

enum _edma_channel_protection_level
eDMA4 channel protection level

Values:

enumerator kEDMA_ChannelProtectionLevelUser
user protection level for eDMA transfers.

enumerator kEDMA_ChannelProtectionLevelPrivileged
Privileged protection level eDMA transfers.

typedef enum *_edma_transfer_size* edma_transfer_size_t
eDMA transfer configuration

typedef enum *_edma_modulo* edma_modulo_t
eDMA modulo configuration

typedef enum *_edma_bandwidth* edma_bandwidth_t
Bandwidth control.

typedef enum *_edma_channel_link_type* edma_channel_link_type_t
Channel link type.

typedef enum *_edma_transfer_type* edma_transfer_type_t
eDMA transfer type

typedef struct *_edma_channel_Preemption_config* edma_channel_Preemption_config_t
eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* edma_minor_offset_config_t
eDMA minor offset configuration

typedef enum *edma_channel_memory_attribute* edma_channel_memory_attribute_t
eDMA channel memory attribute

typedef enum *_edma_channel_swap_size* edma_channel_swap_size_t
eDMA4 channel swap size

typedef enum *_edma_channel_access_type* edma_channel_access_type_t
eDMA4 channel access type

typedef enum *_edma_channel_protection_level* edma_channel_protection_level_t
eDMA4 channel protection level

typedef struct *_edma_channel_config* edma_channel_config_t
eDMA4 channel configuration

typedef *edma_core_tcd_t* edma_tcd_t

eDMA TCD.

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

typedef struct *_edma_transfer_config* edma_transfer_config_t

edma4 channel transfer configuration

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

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

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

---

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

a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0

b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width

c. the totalBytes should align with the bytesEachRequest

d. the srcAddr should align with the srcWidthOfEachTransfer

e. the dstAddr should align with the dstWidthOfEachTransfer

f. the srcAddr should align with srcAddrModulo if modulo feature is enabled

g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.

---

typedef struct *_edma_config* edma_config_t

eDMA global configuration structure.

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

Define callback function for eDMA.

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

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

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

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

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

typedef struct *_edma_handle* edma_handle_t
eDMA transfer handle structure

FSL_EDMA_DRIVER_EDMA4
eDMA driver name

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

DMA_DCHPRI_INDEX(channel)
Compute the offset unit from DCHPRI3.

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

Public Members

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

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

uint8_t channelPriority
Channel priority

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

Public Members

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

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

uint32_t minorOffset
Offset for a minor loop mapping.

struct _edma_channel_config
#include <fsl_edma.h> eDMA4 channel configuration

Public Members

edma_channel_Preemption_config_t channelPreemptionConfig
channel preemption configuration

edma_channel_memory_attribute_t channelReadMemoryAttribute
channel memory read attribute configuration

edma_channel_memory_attribute_t channelWriteMemoryAttribute
channel memory write attribute configuration

edma_channel_swap_size_t channelSwapSize
channel swap size configuration

edma_channel_access_type_t channelAccessType
channel access type configuration

uint8_t channelDataSignExtensionBitPosition
channel data sign extension bit 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 = minorLoopCountsEachMajor-Loop * 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.

2.9 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

2.10 eDMA soc Driver

void EDMA_EnableRequest(DMA_Type *base, dma_request_source_t requestSource)
Enable the eDMA request.

This function enables the eDMA request controlled by SYSCON.

Parameters

- base – eDMA peripheral base address.
- requestSource – eDMA request source, see “dma_request_source_t”.

void EDMA_DisableRequest(DMA_Type *base, dma_request_source_t requestSource)
Disable the eDMA request.

This function disables the eDMA request controlled by SYSCON.

Parameters

- base – eDMA peripheral base address.
- requestSource – eDMA request source, see “dma_request_source_t”.

FSL_EDMA_SOC_DRIVER_VERSION
Driver version 1.0.0.

FSL_EDMA_SOC_IP_DMA3
DMA IP version.

FSL_EDMA_SOC_IP_DMA4

EDMA_CHANNEL_OFFSET

EDMA base address convert macro.

EDMA_CHANNEL_ARRAY_STEP(base)

2.11 Ezhv

enum _arm2ezhv_intctl

Definition of 4 interrupt requests from ARM to EZH-V.

Values:

enumerator kEZHV_ARM2EZHV_MEI

enumerator kEZHV_ARM2EZHV_SEI

enumerator kEZHV_ARM2EZHV_MSI

enumerator kEZHV_ARM2EZHV_MTI

enum _ezhv2arm_int_chan

EZH-V to ARM interrupt channel.

Values:

enumerator kEZHV_EzhvToArmIntChan0

enumerator kEZHV_EzhvToArmIntChan1

enumerator kEZHV_EzhvToArmIntChan2

enumerator kEZHV_EzhvToArmIntChan3

enumerator kEZHV_EzhvToArmIntChan4

enumerator kEZHV_EzhvToArmIntChan5

enumerator kEZHV_EzhvToArmIntChan6

enumerator kEZHV_EzhvToArmIntChan7

enumerator kEZHV_EzhvToArmIntChan8

enumerator kEZHV_EzhvToArmIntChan9

enumerator kEZHV_EzhvToArmIntChan10

enumerator kEZHV_EzhvToArmIntChan11

enumerator kEZHV_EzhvToArmIntChan12

enumerator kEZHV_EzhvToArmIntChan13

enumerator kEZHV_EzhvToArmIntChan14

enumerator kEZHV_EzhvToArmIntChan15

typedef void (*ezhv_callback_t)(void *userData)

define callback function for EZH-V

This callback function is called in EZH-V interrupt handle.

typedef struct *_ezhv_copy_image* ezhv_copy_image_t
Structure for EZH-V copy image to destination address.

Defines source and destination address for copying image with given size.

typedef uint32_t ezhv_api_t
@Brief List of EZH-V APIs

typedef struct *_ezhv_param* ezhv_param_t
parameters used by EZH-V

typedef enum *_arm2ezhv_intctl* arm2ezhv_intctl_t
Definition of 4 interrupt requests from ARM to EZH-V.

typedef enum *_ezhv2arm_int_chan* ezhv2arm_int_chan_t
EZH-V to ARM interrupt channel.

void EZHV_Init(*ezhv_copy_image_t* *ezhvCopyImage)
Initialize the EZH-V.

Parameters

- ezhvCopyImage – The information about the EZH-V image to copy.

void EZHV_InitWithoutFirmware(void)
Initialize the EZH-V.

This function is similar with EZHV_Init, the difference is this function does not install the firmware, the firmware could be installed using EZHV_InstallFirmware.

void EZHV_InstallFirmware(*ezhv_copy_image_t* *ezhvCopyImage)
install EZH-V firmware by given image info

Parameters

- ezhvCopyImage – The information about the EZH-V image to copy.

void EZHV_Boot(uint32_t bootAddr)
Boot EZH-V from given address bootAddr.

Parameters

- bootAddr – The boot address.

void EZHV_Deinit(void)
Deinitialize the EZH-V.

void EZHV_SetCallback(*ezhv_callback_t* callback, uint16_t channel, void *userData)
Install the EZH-V callback function.

Parameters

- callback – EZH-V callback function pointer.
- channel – interrupt channel index.
- userData – A parameter for the callback function.

void EZHV_SetPara(*ezhv_param_t* *para)
Set the parameter used by EZH-V.

Parameters

- para – Parameter written into shared mem between ARM and EZH-V.

uint32_t *EZHV_GetParaAddr(void)

Get shared space address.

Returns

The start address of shared space

void EZHV_WakeUpEzhv(*arm2ezhv_intctl_t* arm2ezhvInt)

Check the wait status and wake up EZH-V.

Parameters

- arm2ezhvInt – The interrupt request which will send to the EZH-V.

static inline void EZHV_EnableArm2EzhvInt(*arm2ezhv_intctl_t* arm2ezhvInt)

Enable interrupt request from ARM core to EZH-V core.

Note: This interrupt can be used to send interrupt request to EZH-V

Parameters

- arm2ezhvInt – The interrupt request which will send to the EZH-V.

static inline void EZHV_DisableArm2EzhvInt(*arm2ezhv_intctl_t* arm2ezhvInt)

Disable interrupt request from ARM core to EZH-V core.

Parameters

- arm2ezhvInt – The interrupt request type.

void EZHV_DriverIRQHandler(void)

EZH-V IRQ handler.

static inline void EZHV_EnableEzhv2ArmIntChan(*ezhv2arm_int_chan_t* chan)

Enable the EZH-V interrupt channel to ARM core.

Parameters

- chan – Ezhv to arm interrupt channel index.

static inline void EZHV_DisableEzhv2ArmIntChan(*ezhv2arm_int_chan_t* chan)

Disable the EZH-V interrupt channel to ARM core.

Parameters

- chan – Ezhv to arm interrupt channel index.

static inline uint32_t EZHV_GetEzhv2ArmIntChan(void)

Get EZH-V interrupt outputs.

static inline void EZHV_ClearEzhv2ArmIntChan(*ezhv2arm_int_chan_t* chan)

Clear EZH-V interrupt outputs.

Parameters

- chan – EZH-V to ARM interrupt channel index.

static inline bool EZHV_GetEzhvStopStatusFlag(void)

Get EZH-V stop status flag.

static inline bool EZHV_GetEzhvHaltStatusFlag(void)

Get EZH-V halt status flag.

static inline bool EZHV_GetEzhvWaitStatusFlag(void)

Get EZH-V wait status flag.

static inline bool EZHV_GetEzhvWakeupStatusFlag(void)
Get EZH-V wakeup status flag.

FSL_EZHV_DRIVER_VERSION
cache driver version.

EZHV_API_MAX_ARG

uint32_t srcAddr

uint32_t destAddr

uint32_t size

ezhv_api_t ezhvApi

uint32_t argc

uint32_t *argv[8U]

EZHV_M_EXT_INT
EZHV inside machine-mode external isr offset.

EZHV_SHARED_DATA_ADDR
shared data space between ARM and EZHV, space size is 256 Bytes

EZHV_INT_CHAN_NUM
definition for active ezhv2arm int channel number

please redefine this macro's value if less channels are enabled

struct _ezhv_copy_image
#include <fsl_ezhv.h> Structure for EZH-V copy image to destination address.

Defines source and destination address for copying image with given size.

struct _ezhv_param
#include <fsl_ezhv.h> parameters used by EZH-V

2.12 FGPIO Driver

2.13 FlexIO: FlexIO Driver

2.14 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

2.15 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

2.16 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

2.17 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

2.18 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.

2.19 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

2.20 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,
```

(continues on next page)

(continued from previous page)

```
.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,
```

(continues on next page)

(continued from previous page)

```
.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.

2.21 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,
```

(continues on next page)

(continued from previous page)

```
.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 ___unnamed71___

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.

2.22 FREQME: Frequency Measurement

2.23 GDET

status_t GDET_Init(GDET_Type *base)

Initialize GDET.

This function initializes GDET block and setting.

Parameters

- base – GDET peripheral base address

Returns

Status of the init operation

void GDET_Deinit(GDET_Type *base)

Deinitialize GDET.

This function deinitializes GDET secure counter.

Parameters

- base – GDET peripheral base address

status_t GDET_Enable(GDET_Type *base)

Enable GDET.

This function enables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns

Status of the enable operation

status_t GDET_Disable(GDET_Type *base)

Disable GDET.

This function disables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns

Status of the disable operation

status_t GDET_IsolateOn(GDET_Type *base)

Turn on GDET isolation.

This function turns on isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns

Status of the operation

status_t GDET_IsolateOff(GDET_Type *base)

Turn off GDET isolation.

This function turns off isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns
Status of the operation

status_t GDET_ReconfigureVoltageMode(GDET_Type *base, *gdet_core_voltage_t* voltage)

Change expected core voltage.

This function changes core voltage which Glitch detector expect.

Parameters

- base – GDET peripheral base address
- voltage – Expected core voltage

Returns
Status of the GDET reconfiguration operation

FSL_GDET_DRIVER_VERSION

Defines GDET driver version 2.1.0.

Change log:

- Version 2.1.0
  - Update for multiple instances
  - Fix bug in isolation off API
  - Add enable and disable APIs
- 2.0.1
  - Fix MISRA in GDET_ReconfigureVoltageMode()
- Version 2.0.0
  - initial version

typedef uint32_t gdet_core_voltage_t

GDET Core Voltage.

These constants are used to define core voltage argument to be used with GDET_ReconfigureVoltageMode(). Different SoC may support various volatages, refer to documentation.

void GDET_DriverIRQHandler(void)

kGDET_0_9v

Voltage (0.9V)

kGDET_1_0v

Voltage (1.0V)

kGDET_1_1v

Voltage (1.1V)

2.24 Glitch Detect

2.25 GLIKEY

2.26 GLIKEY

Values:

enumerator kStatus_GLIKEY_LockedError
GLIKEY status for locked SFR registers (unexpected) .

enumerator kStatus_GLIKEY_NotLocked
GLIKEY status for unlocked SFR registers.

enumerator kStatus_GLIKEY_Locked
GLIKEY status for locked SFR registers.

enumerator kStatus_GLIKEY_DisabledError
GLIKEY status for disabled error.

FSL_GLIKEY_DRIVER_VERSION
Defines GLIKEY driver version 2.0.1.

Change log:

- Version 2.0.1
  - Implement INIT state recovery from the LOCKED state after a reset when the previous index was locked.
- Version 2.0.0
  - Initial version

GLIKEY_CODEWORD_STEP1

GLIKEY_CODEWORD_STEP2

GLIKEY_CODEWORD_STEP3

GLIKEY_CODEWORD_STEP4

GLIKEY_CODEWORD_STEP5

GLIKEY_CODEWORD_STEP6

GLIKEY_CODEWORD_STEP7

GLIKEY_CODEWORD_STEP_EN

GLIKEY_FSM_WR_DIS

GLIKEY_FSM_INIT

GLIKEY_FSM_STEP1

GLIKEY_FSM_STEP2

GLIKEY_FSM_STEP3

GLIKEY_FSM_STEP4

GLIKEY_FSM_LOCKED

GLIKEY_FSM_WR_EN

GLIKEY_FSM_SSR_RESET

uint32_t GLIKEY_GetStatus(GLIKEY_Type *base)

Retreives the current status of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Glikey status information

status_t GLIKEY_IsLocked(GLIKEY_Type *base)

Get if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_Locked – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status

status_t GLIKEY_CheckLock(GLIKEY_Type *base)

Check if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_LockedError – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status kStatus_Success if success

status_t GLIKEY_SyncReset(GLIKEY_Type *base)

Perform a synchronous reset of Glikey.

This function performs a synchrounous reset of the Glikey. This results in:

- Glikey will return to the INIT state, unless it is in the LOCK state

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntEnable(GLIKEY_Type *base, uint32_t value)

Set interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[in]** Value to set the interrupt enable flag to, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_GetIntEnable(GLIKEY_Type *base, uint32_t *value)

Get interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[out]** Pointer which will be filled with the interrupt enable status, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success

status_t GLIKEY_ClearIntStatus(GLIKEY_Type *base)

Clear the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntStatus(GLIKEY_Type *base)

Set the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_Lock(GLIKEY_Type *base)

Lock Glikey SFR (Special Function Registers) interface.

This operation locks the Glikey SFR interface if it is not locked yet.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success

status_t GLIKEY_LockIndex(GLIKEY_Type *base)

Lock Glikey index.

This operation is used to lock a Glikey index. It can only be executed from the WR_EN state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchrous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_IsIndexLocked(GLIKEY_Type *base, uint32_t index)

Check if Glikey index is locked.

This operation returns the locking status of Glikey index.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns
kStatus_GLIKEY_Locked if locked, kStatus_GLIKEY_NotLocked if unlocked Possible errors: kStatus_Fail

status_t GLIKEY_StartEnable(GLIKEY_Type *base, uint32_t index)

Start Glikey enable.

This operation is used to set a new index and start a the sequence to enable it. It needs to be started from the INIT state. If the new index is already locked Glikey will go to LOCKED state, otherwise it will go to STEP1 state. If this operation is used when Glikey is in any state other than INIT Glikey will go to WR_DIS state. It can only recover from this state through a reset (synchrounous or asyncrhonous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns
Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_Fail

status_t GLIKEY_ContinueEnable(GLIKEY_Type *base, uint32_t codeword)

Continue Glikey enable.

This operation is used to progress through the different states of the state machine, starting from STEP1 until the state WR_EN is reached. Each next state of the state machine can only be reached by providing the right codeword to this function. If anything goes wrong the state machine will go to WR_DIS state and can only recover from it through a reset (synchrous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- codeword – **[in]** Encoded word for progressing to next FSM state (see GLIKEY_CODEWORD_STEPx/EN)

Returns
Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_Fail, kStatus_GLIKEY_DisabledError

status_t GLIKEY_EndOperation(GLIKEY_Type *base)

End Glikey operation.

This operation is used to end a Glikey operation. It can only be executed from the WR_EN, LOCKED and RESET states. Executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchrous or asynchronous) to go back to INIT state. After this operation Glikey will go to INIT state or stay in LOCKED state when the index was locked. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns
A code-flow protected error code (see nxpCsslFlowProtection)

Returns
Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_ResetIndex(GLIKEY_Type *base, uint32_t index)

Reset Glikey index.

This operation is used to reset a Glikey index. It can only be executed from the INIT state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchrous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active or the index is locked this operation will return an error.

Returns
A code-flow protected error code (see nxpCsslFlowProtection)

Returns
Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

2.27 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION

GPIO driver version.

enum _gpio_pin_direction

GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput

Set current pin as digital input

enumerator kGPIO_DigitalOutput

Set current pin as digital output

enum _gpio_checker_attribute

GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW

User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW

User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW

User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW

User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW

User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW

User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO_IgnoreAttributeCheck
Ignores the attribute check

enum _gpio_interrupt_config
Configures the interrupt generation condition.

Values:

enumerator kGPIO_InterruptStatusFlagDisabled
Interrupt status flag is disabled.

enumerator kGPIO_DMARisingEdge
ISF flag and DMA request on rising edge.

enumerator kGPIO_DMAFallingEdge
ISF flag and DMA request on falling edge.

enumerator kGPIO_DMAEitherEdge
ISF flag and DMA request on either edge.

enumerator kGPIO_FlagRisingEdge
Flag sets on rising edge.

enumerator kGPIO_FlagFallingEdge
Flag sets on falling edge.

enumerator kGPIO_FlagEitherEdge
Flag sets on either edge.

enumerator kGPIO_InterruptLogicZero
Interrupt when logic zero.

enumerator kGPIO_InterruptRisingEdge
Interrupt on rising edge.

enumerator kGPIO_InterruptFallingEdge
Interrupt on falling edge.

enumerator kGPIO_InterruptEitherEdge
Interrupt on either edge.

enumerator kGPIO_InterruptLogicOne
Interrupt when logic one.

enumerator kGPIO_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kGPIO_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

enum _gpio_interrupt_selection
Configures the selection of interrupt/DMA request/trigger output.

Values:

enumerator kGPIO_InterruptOutput0
Interrupt/DMA request/trigger output 0.

enumerator kGPIO_InterruptOutput1
Interrupt/DMA request/trigger output 1.

enum gpio_pin_interrupt_control_t
GPIO pin and interrupt control.

Values:

enumerator kGPIO_PinControlNonSecure
Pin Control Non-Secure.

enumerator kGPIO_InterruptControlNonSecure
Interrupt Control Non-Secure.

enumerator kGPIO_PinControlNonPrivilege
Pin Control Non-Privilege.

enumerator kGPIO_InterruptControlNonPrivilege
Interrupt Control Non-Privilege.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_checker_attribute* gpio_checker_attribute_t
GPIO checker attribute.

typedef struct *_gpio_pin_config* gpio_pin_config_t
The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

typedef enum *_gpio_interrupt_config* gpio_interrupt_config_t
Configures the interrupt generation condition.

typedef enum *_gpio_interrupt_selection* gpio_interrupt_selection_t
Configures the selection of interrupt/DMA request/trigger output.

typedef struct *_gpio_version_info* gpio_version_info_t
GPIO version information.

GPIO_FIT_REG(value)

struct _gpio_pin_config
#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Public Members

gpio_pin_direction_t pinDirection
GPIO direction, input or output

uint8_t outputLogic
Set a default output logic, which has no use in input

struct _gpio_version_info
#include <fsl_gpio.h> GPIO version information.

Public Members

uint16_t feature
: Feature Specification Number.

uint8_t minor
: Minor Version Number.

uint8_t major
: Major Version Number.

2.28 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)
: Initializes the GPIO peripheral.

  This function ungates the GPIO clock.

  Parameters

  - base – GPIO peripheral base pointer.

void GPIO_PortDenit(GPIO_Type *base)
: Denitializes the GPIO peripheral.

  Parameters

  - base – GPIO peripheral base pointer.

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)
: Initializes a GPIO pin used by the board.

  To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

  This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalOutput,
  0,
}
```

  Parameters

  - base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
  - pin – GPIO port pin number
  - config – GPIO pin configuration pointer

void GPIO_GetVersionInfo(GPIO_Type *base, *gpio_version_info_t* *info)
: Get GPIO version information.

  Parameters

  - base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

- info – GPIO version information

static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, *gpio_pin_interrupt_control_t* mask)

lock or unlock secure privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Disable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Disable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)

Enable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)

Disable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Enable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)

Disable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputEnable(GPIO_Type *base, uint32_t mask)

Enable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputDisable(GPIO_Type *base, uint32_t mask)

Disable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_config_t* config)

Configures the gpio pin interrupt/DMA request.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- config – GPIO pin interrupt configuration.
  - kGPIO_InterruptStatusFlagDisabled: Interrupt/DMA request disabled.
  - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kGPIO_InterruptLogicZero : Interrupt when logic zero.
  - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
  - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
  - kGPIO_InterruptEitherEdge : Interrupt on either edge.
  - kGPIO_InterruptLogicOne : Interrupt when logic one.
  - kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
  - kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

static inline void GPIO_SetPinInterruptChannel(GPIO_Type *base, uint32_t pin, *gpio_interrupt_selection_t* selection)

Configures the gpio pin interrupt/DMA request/trigger output channel selection.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- selection – GPIO pin interrupt output selection.

- kGPIO_InterruptOutput0: Interrupt/DMA request/trigger output 0.
- kGPIO_InterruptOutput1 : Interrupt/DMA request/trigger output 1.

uint32_t GPIO_GpioGetInterruptFlags(GPIO_Type *base)

Read the GPIO interrupt status flags.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)

Returns

The current GPIO's interrupt status flag. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint32_t GPIO_GpioGetInterruptChannelFlags(GPIO_Type *base, uint32_t channel)

Read the GPIO interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

Returns

The current GPIO's interrupt status flag based on the selected interrupt channel. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint8_t GPIO_PinGetInterruptFlag(GPIO_Type *base, uint32_t pin)

Read individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on)
- pin – GPIO specific pin number.

Returns

The current selected pin's interrupt status flag.

void GPIO_GpioClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

void GPIO_GpioClearInterruptChannelFlags(GPIO_Type *base, uint32_t mask, uint32_t channel)

Clears GPIO pin interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

void GPIO_PinClearInterruptFlag(GPIO_Type *base, uint32_t pin)

Clear GPIO individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on).
- pin – GPIO specific pin number.

static inline void GPIO_SetMultipleInterruptPinsConfig(GPIO_Type *base, uint32_t mask, *gpio_interrupt_config_t* config)

Sets the GPIO interrupt configuration in PCR register for multiple pins.

Parameters

- base – GPIO peripheral base pointer.
- mask – GPIO pin number macro.
- config – GPIO pin interrupt configuration.
  - kGPIO_InterruptStatusFlagDisabled: Interrupt disabled.
  - kGPIO_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).
  - kGPIO_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).
  - kGPIO_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).
  - kGPIO_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).
  - kGPIO_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).
  - kGPIO_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).
  - kGPIO_InterruptLogicZero : Interrupt when logic zero.
  - kGPIO_InterruptRisingEdge : Interrupt on rising edge.
  - kGPIO_InterruptFallingEdge: Interrupt on falling edge.
  - kGPIO_InterruptEitherEdge : Interrupt on either edge.
  - kGPIO_InterruptLogicOne : Interrupt when logic one.
  - kGPIO_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).
  - kGPIO_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit)..

void GPIO_CheckAttributeBytes(GPIO_Type *base, *gpio_checker_attribute_t* attribute)

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

2.29 I3C: I3C Driver

FSL_I3C_DRIVER_VERSION

I3C driver version.

I3C status return codes.

Values:

enumerator kStatus_I3C_Busy
The master is already performing a transfer.

enumerator kStatus_I3C_Idle
The slave driver is idle.

enumerator kStatus_I3C_Nak
The slave device sent a NAK in response to an address.

enumerator kStatus_I3C_WriteAbort
The slave device sent a NAK in response to a write.

enumerator kStatus_I3C_Term
The master terminates slave read.

enumerator kStatus_I3C_HdrParityError
Parity error from DDR read.

enumerator kStatus_I3C_CrcError
CRC error from DDR read.

enumerator kStatus_I3C_ReadFifoError
Read from M/SRDATAB register when FIFO empty.

enumerator kStatus_I3C_WriteFifoError
Write to M/SWDATAB register when FIFO full.

enumerator kStatus_I3C_MsgError
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus_I3C_InvalidReq
Invalid use of request.

enumerator kStatus_I3C_Timeout
The module has stalled too long in a frame.

enumerator kStatus_I3C_SlaveCountExceed
The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

enumerator kStatus_I3C_IBIWon
The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus_I3C_OverrunError
Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus_I3C_UnderrunError
Slave internal to-bus buffer/FIFO underrun

enumerator kStatus_I3C_UnderrunNak
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus_I3C_InvalidStart
Slave invalid start flag

enumerator kStatus_I3C_SdrParityError
SDR parity error

enumerator kStatus_I3C_S0S1Error
S0 or S1 error

enum _i3c_hdr_mode
I3C HDR modes.

Values:

enumerator kI3C_HDRModeNone

enumerator kI3C_HDRModeDDR

enumerator kI3C_HDRModeTSP

enumerator kI3C_HDRModeTSL

typedef enum *_i3c_hdr_mode* i3c_hdr_mode_t
I3C HDR modes.

typedef struct *_i3c_device_info* i3c_device_info_t
I3C device information.

I3C_RETRY_TIMES
Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C_MAX_DEVCNT

I3C_IBI_BUFF_SIZE

struct _i3c_device_info
#include <fsl_i3c.h> I3C device information.

Public Members

uint8_t dynamicAddr
Device dynamic address.

uint8_t staticAddr
Static address.

uint8_t dcr
Device characteristics register information.

uint8_t bcr
Bus characteristics register information.

uint16_t vendorID
Device vendor ID(manufacture ID).

uint32_t partNumber
Device part number info

uint16_t maxReadLength
Maximum read length.

uint16_t maxWriteLength
Maximum write length.

uint8_t hdrMode
Support hdr mode, could be OR logic in i3c_hdr_mode.

2.30 I3C Common Driver

typedef struct *_i3c_config* i3c_config_t

Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

uint32_t I3C_GetInstance(I3C_Type *base)

Get which instance current I3C is used.

Parameters

- base – The I3C peripheral base address.

void I3C_GetDefaultConfig(*i3c_config_t* *config)

Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster                = kI3C_MasterCapable;
config->disableTimeout              = false;
config->hKeep                       = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop         = true;
config->enableOpenDrainHigh         = true;
config->baudRate_Hz.i2cBaud         = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress        = 0x0AU;
config->slowClock_Hz                = 1000000U;
config->enableSlave                 = true;
config->vendorID                    = 0x11BU;
config->enableRandomPart            = false;
config->partNumber                  = 0;
config->dcr                         = 0;
config->bcr = 0;
config->hdrMode             = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest       = false;
config->ignoreS0S1Error     = false;
config->offline             = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C_Init().

Parameters

- config – **[out]** User provided configuration structure for default values. Refer to i3c_config_t.

void I3C_Init(I3C_Type *base, const *i3c_config_t* *config, uint32_t sourceClock_Hz)

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.

- config – User provided peripheral configuration. Use I3C_GetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

struct _i3c_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

uint8_t masterDynamicAddress

Main master dynamic address configuration.

uint32_t maxWriteLength

Maximum write length.

uint32_t maxReadLength

Maximum read length.

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint8_t hdrMode
: Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest
: Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error
: Whether to ignore S0/S1 error in SDR mode.

bool offline
: Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop
: Whether to assert start/stop status only the time slave is addressed.

2.31 I3C Master Driver

void I3C_MasterGetDefaultConfig(*i3c_master_config_t* *masterConfig)

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster            = kI3C_MasterOn;
masterConfig->disableTimeout          = false;
masterConfig->hKeep                   = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop     = true;
masterConfig->enableOpenDrainHigh     = true;
masterConfig->baudRate_Hz             = 100000U;
masterConfig->busType                 = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with I3C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to i3c_master_config_t.

void I3C_MasterInit(I3C_Type *base, const *i3c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- masterConfig – User provided peripheral configuration. Use I3C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void I3C_MasterDeinit(I3C_Type *base)

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The I3C peripheral base address.

status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)

status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)

status_t I3C_CheckForBusyBus(I3C_Type *base)

static inline void I3C_MasterEnable(I3C_Type *base, *i3c_master_enable_t* enable)

Set I3C module master mode.

Parameters

- base – The I3C peripheral base address.
- enable – Enable master mode.

void I3C_SlaveGetDefaultConfig(*i3c_slave_config_t* *slaveConfig)

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave              = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

Parameters

- slaveConfig – **[out]** User provided configuration structure for default values. Refer to i3c_slave_config_t.

void I3C_SlaveInit(I3C_Type *base, const *i3c_slave_config_t* *slaveConfig, uint32_t slowClock_Hz)

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- slowClock_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH defines as 1, this parameter is useless.

void I3C_SlaveDeinit(I3C_Type *base)

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnable)

Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnable – Enable or disable.

static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_master_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C master status flag state.

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag
- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

Attempts to clear other flags has no effect.

See also:

_i3c_master_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _i3c_master_flags enumerators OR'd together. You may pass the result of a previous call to I3C_MasterGetStatusFlags().

static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_master_error_flags

Parameters

- base – The I3C peripheral base address.

Returns
State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C master error status flag state.

See also:

_i3c_master_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of _i3c_master_error_flags enumerators OR'd together. You may pass the result of a previous call to I3C_MasterGetStatusFlags().

i3c_master_state_t I3C_MasterGetState(I3C_Type *base)

Gets the I3C master state.

Parameters

- base – The I3C peripheral base address.

Returns
I3C master state.

static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_flags

Parameters

- base – The I3C peripheral base address.

Returns
State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Attempts to clear other flags has no effect.

See also:

_i3c_slave_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _i3c_slave_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_error_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave error status flag state.

See also:

_i3c_slave_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of _i3c_slave_error_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)

Gets the I3C slave state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C slave activity state, refer i3c_slave_activity_state_t.

status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)

static inline void I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C master interrupt requests.

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _i3c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C master interrupt requests.

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _i3c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_MasterGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of _i3c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of _i3c_master_flags enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag

- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C master DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C master transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C master receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

static inline void I3C_MasterSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C master FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The kI3C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The kI3C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C master FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The kI3C_SlaveTxReadyFlag flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The kI3C_SlaveRxReadyFlag flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

void I3C_MasterSetBaudRate(I3C_Type *base, const *i3c_baudrate_hz_t* *baudRate_Hz, uint32_t sourceClock_Hz)

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

- base – The I3C peripheral base address.
- baudRate_Hz – Pointer to structure of requested bus frequency in Hertz.
- sourceClock_Hz – I3C functional clock frequency in Hertz.

static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The I3C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t I3C_MasterStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the a address parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- rxSize – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.

status_t I3C_MasterStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.

status_t I3C_MasterRepeatedStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like I3C_MasterStart(), it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the rxSize = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the rxSize configuration.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- rxSize – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.

static inline *status_t* I3C_MasterRepeatedStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like I3C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.

status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)

Performs a polling send transfer on the I2C/I3C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_I3C_Nak.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- flags – Bit mask of options for the transfer. See enumeration _i3c_master_transfer_flags for available options.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Nak – The slave device sent a NAK in response to an address.
- kStatus_I3C_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_WriteFifoError – Write to M/SWDATAB register when FIFO full.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_MasterReceive(I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- flags – Bit mask of options for the transfer. See enumeration _i3c_master_transfer_flags for available options.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Term – The master terminates slave read.
- kStatus_I3C_HdrParityError – Parity error from DDR read.
- kStatus_I3C_CrcError – CRC error from DDR read.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_ReadFifoError – Read from M/SRDATAB register when FIFO empty.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_MasterStop(I3C_Type *base)

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The I3C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_InvalidReq – Invalid use of request.

void I3C_MasterEmitRequest(I3C_Type *base, *i3c_bus_request_t* masterReq)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- masterReq – I3C master request of type i3c_bus_request_t

static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, *i3c_ibi_response_t* ibiResponse)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- ibiResponse – I3C master emit IBI response of type i3c_ibi_response_t

void I3C_MasterRegisterIBI(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master register IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to ibi rule description of type i3c_register_ibi_addr_t

void I3C_MasterGetIBIRules(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master get IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to store the read out ibi rule description.

i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)

I3C master get IBI Type.

Parameters

- base – The I3C peripheral base address.

Return values

i3c_ibi_type_t – Type of i3c_ibi_type_t.

static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)

I3C master get IBI Address.

Parameters

- base – The I3C peripheral base address.

Return values

The – 8-bit IBI address.

status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t count, *i3c_master_daa_baudrate_t* *daaBaudRate)

Performs a DAA in the i3c bus with specified temporary baud rate.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list.
- daaBaudRate – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- kStatus_I3C_SlaveCountExceed – The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

static inline *status_t* I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t count)

Performs a DAA in the i3c bus.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- kStatus_I3C_SlaveCountExceed – The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)

Get device information list after DAA process is done.

Parameters

- base – The I3C peripheral base address.
- count – **[out]** The pointer to store the available device count.

Returns

Pointer to the i3c_device_info_t array.

void I3C_MasterClearDeviceCount(I3C_Type *base)

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

Parameters

- base – The I3C peripheral base address.

status_t I3C_MasterTransferBlocking(I3C_Type *base, *i3c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C/I3C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The I3C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_IBIWon – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Nak – The slave device sent a NAK in response to an address.
- kStatus_I3C_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus_I3C_Term – The master terminates slave read.
- kStatus_I3C_HdrParityError – Parity error from DDR read.

- kStatus_I3C_CrcError – CRC error from DDR read.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_ReadFifoError – Read from M/SRDATAB register when FIFO empty.
- kStatus_I3C_WriteFifoError – Write to M/SWDATAB register when FIFO full.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

void I3C_MasterTransferCreateHandle(I3C_Type *base, *i3c_master_handle_t* *handle, const *i3c_master_transfer_callback_t* *callback, void *userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_MasterTransferNonBlocking(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I3C_MasterTransferGetCount(I3C_Type *base, *i3c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void I3C_MasterTransferAbort(I3C_Type *base, *i3c_master_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to the I3C master driver handle.

enum _i3c_master_flags

I3C master peripheral flags.

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag

- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterBetweenFlag
Between messages/DAAs flag

enumerator kI3C_MasterNackDetectFlag
NACK detected flag

enumerator kI3C_MasterSlaveStartFlag
Slave request start flag

enumerator kI3C_MasterControlDoneFlag
Master request complete flag

enumerator kI3C_MasterCompleteFlag
Transfer complete flag

enumerator kI3C_MasterRxReadyFlag
Rx data ready in Rx buffer flag

enumerator kI3C_MasterTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_MasterArbitrationWonFlag
Header address won arbitration flag

enumerator kI3C_MasterErrorFlag
Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag
Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum __i3c_master_error_flags
I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag
Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag
Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag
Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag
CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag
Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag
Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag
Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag
The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags
All error flags

enum _i3c_master_state
I3C working master state.

Values:

enumerator kI3C_MasterStateIdle
Bus stopped.

enumerator kI3C_MasterStateSlvReq
Bus stopped but slave holding SDA low.

enumerator kI3C_MasterStateMsgSdr
In SDR Message mode from using MWMSG_SDR.

enumerator kI3C_MasterStateNormAct
In normal active SDR mode.

enumerator kI3C_MasterStateDdr
In DDR Message mode.

enumerator kI3C_MasterStateDaa
In ENTDAA mode.

enumerator kI3C_MasterStateIbiAck
Waiting on IBI ACK/NACK decision.

enumerator kI3C_MasterStateIbiRcv
Receiving IBI.

enum _i3c_master_enable
I3C master enable configuration.

Values:

enumerator kI3C_MasterOff
Master off.

enumerator kI3C_MasterOn
Master on.

enumerator kI3C_MasterCapable
Master capable.

enum _i3c_master_hkeep

I3C high keeper configuration.

Values:

enumerator kI3C_MasterHighKeeperNone

Use PUR to hold SCL high.

enumerator kI3C_MasterHighKeeperWiredIn

Use pin_HK controls.

enumerator kI3C_MasterPassiveSDA

Hi-Z for Bus Free and hold SDA.

enumerator kI3C_MasterPassiveSDASCL

Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum _i3c_bus_request

Emits the requested operation when doing in pieces vs. by message.

Values:

enumerator kI3C_RequestNone

No request.

enumerator kI3C_RequestEmitStartAddr

Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop

Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack

Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA

Process DAA.

enumerator kI3C_RequestForceExit

Request to force exit.

enumerator kI3C_RequestAutoIbi

Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type

Bus type with EmitStartAddr.

Values:

enumerator kI3C_TypeI3CSdr

SDR mode of I3C.

enumerator kI3C_TypeI2C

Standard i2c protocol.

enumerator kI3C_TypeI3CDdr

HDR-DDR mode of I3C.

enum _i3c_ibi_response

IBI response.

Values:

enumerator kI3C_IbiRespAck

ACK with no mandatory byte.

enumerator kI3C_IbiRespNack
NACK.

enumerator kI3C_IbiRespAckMandatory
ACK with mandatory byte.

enumerator kI3C_IbiRespManual
Reserved.

enum _i3c_ibi_type
IBI type.

Values:

enumerator kI3C_IbiNormal
In-band interrupt.

enumerator kI3C_IbiHotJoin
slave hot join.

enumerator kI3C_IbiMasterRequest
slave master ship request.

enum _i3c_ibi_state
IBI state.

Values:

enumerator kI3C_IbiReady
In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed
In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending
In-band interrupt Ack/Nack pending for decision.

enum _i3c_direction
Direction of master and slave transfers.

Values:

enumerator kI3C_Write
Master transmit.

enumerator kI3C_Read
Master receive.

enum _i3c_tx_trigger_level
Watermark of TX int/dma trigger level.

Values:

enumerator kI3C_TxTriggerOnEmpty
Trigger on empty.

enumerator kI3C_TxTriggerUntilOneQuarterOrLess
Trigger on 1/4 full or less.

enumerator kI3C_TxTriggerUntilOneHalfOrLess
Trigger on 1/2 full or less.

enumerator kI3C_TxTriggerUntilOneLessThanFull
Trigger on 1 less than full or less.

enum _i3c_rx_trigger_level

Watermark of RX int/dma trigger level.

Values:

enumerator kI3C_RxTriggerOnNotEmpty

Trigger on not empty.

enumerator kI3C_RxTriggerUntilOneQuarterOrMore

Trigger on 1/4 full or more.

enumerator kI3C_RxTriggerUntilOneHalfOrMore

Trigger on 1/2 full or more.

enumerator kI3C_RxTriggerUntilThreeQuarterOrMore

Trigger on 3/4 full or more.

enum _i3c_rx_term_ops

I3C master read termination operations.

Values:

enumerator kI3C_RxTermDisable

Master doesn't terminate read, used for CCC transfer.

enumerator kI3C_RxAutoTerm

Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C_RxTermLastByte

Master terminates read at any time after START, no length limitation.

enum _i3c_start_scl_delay

I3C start SCL delay options.

Values:

enumerator kI3C_NoDelay

No delay.

enumerator kI3C_IncreaseSclHalfPeriod

Increases SCL clock period by 1/2.

enumerator kI3C_IncreaseSclOnePeriod

Increases SCL clock period by 1.

enumerator kI3C_IncreaseSclOneAndHalfPeriod

Increases SCL clock period by 1 1/2

enum _i3c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _i3c_master_transfer::flags field.

Values:

enumerator kI3C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kI3C_TransferNoStopFlag
Don't send a stop condition.

enumerator kI3C_TransferWordsFlag
Transfer in words, else transfer in bytes.

enumerator kI3C_TransferDisableRxTermFlag
Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C_TransferRxAutoTermFlag
Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C_MasterReceive.

enumerator kI3C_TransferStartWithBroadcastAddr
Start transfer with 0x7E, then read/write data with device address.

typedef enum *_i3c_master_state* i3c_master_state_t
I3C working master state.

typedef enum *_i3c_master_enable* i3c_master_enable_t
I3C master enable configuration.

typedef enum *_i3c_master_hkeep* i3c_master_hkeep_t
I3C high keeper configuration.

typedef enum *_i3c_bus_request* i3c_bus_request_t
Emits the requested operation when doing in pieces vs. by message.

typedef enum *_i3c_bus_type* i3c_bus_type_t
Bus type with EmitStartAddr.

typedef enum *_i3c_ibi_response* i3c_ibi_response_t
IBI response.

typedef enum *_i3c_ibi_type* i3c_ibi_type_t
IBI type.

typedef enum *_i3c_ibi_state* i3c_ibi_state_t
IBI state.

typedef enum *_i3c_direction* i3c_direction_t
Direction of master and slave transfers.

typedef enum *_i3c_tx_trigger_level* i3c_tx_trigger_level_t
Watermark of TX int/dma trigger level.

typedef enum *_i3c_rx_trigger_level* i3c_rx_trigger_level_t
Watermark of RX int/dma trigger level.

typedef enum *_i3c_rx_term_ops* i3c_rx_term_ops_t
I3C master read termination operations.

typedef enum *_i3c_start_scl_delay* i3c_start_scl_delay_t
I3C start SCL delay options.

typedef struct *_i3c_register_ibi_addr* i3c_register_ibi_addr_t
Structure with setting master IBI rules and slave registry.

typedef struct *_i3c_baudrate* i3c_baudrate_hz_t
Structure with I3C baudrate settings.

typedef struct *_i3c_master_daa_baudrate* i3c_master_daa_baudrate_t

I3C DAA baud rate configuration.

typedef struct *_i3c_master_config* i3c_master_config_t

Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef struct *_i3c_master_transfer* i3c_master_transfer_t

typedef struct *_i3c_master_handle* i3c_master_handle_t

typedef struct *_i3c_master_transfer_callback* i3c_master_transfer_callback_t

i3c master callback functions.

typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)

Typedef for master interrupt handler.

struct _i3c_register_ibi_addr

#include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.

Public Members

uint8_t address[5]

Address array for registry.

bool i3cFastStart

Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct _i3c_baudrate

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

uint32_t i2cBaud

Desired I2C baud rate in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct _i3c_master_daa_baudrate

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

uint32_t sourceClock_Hz

FCLK, function clock in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct __i3c_master_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz

Desired baud rate settings.

struct __i3c_master_transfer_callback

#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)

Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)

IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Transfer complete callback

struct __i3c_master_transfer

#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
: Bit mask of options for the transfer. See enumeration _i3c_master_transfer_flags for available options. Set to 0 or kI3C_TransferDefaultFlag for normal transfers.

uint8_t slaveAddress
: The 7-bit slave address.

i3c_direction_t direction
: Either kI3C_Read or kI3C_Write.

uint32_t subaddress
: Sub address. Transferred MSB first.

size_t subaddressSize
: Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
: Pointer to data to transfer.

size_t dataSize
: Number of bytes to transfer.

i3c_bus_type_t busType
: bus type.

i3c_ibi_response_t ibiResponse
: ibi response during transfer.

struct __i3c_master_handle
: *#include <fsl_i3c.h>* Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
: Transfer state machine current state.

uint32_t remainingBytes
: Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps
: Read termination operation.

i3c_master_transfer_t transfer
: Copy of the current transfer info.

uint8_t ibiAddress
: Slave address which request IBI.

uint8_t *ibiBuff
: Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
: IBI payload size.

i3c_ibi_type_t ibiType
: IBI type.

i3c_master_transfer_callback_t callback
Callback functions pointer.

void *userData
Application data passed to callback.

2.32 I3C Master DMA Driver

void I3C_MasterTransferCreateHandleEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle, const *i3c_master_edma_callback_t* *callback, void *userData, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle)

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_MasterTransferAbortDMA() API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.
- rxDmaHandle – Handle for the DMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

status_t I3C_MasterTransferEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t I3C_MasterTransferGetCountEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.

- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

void I3C_MasterTransferAbortEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- i3cHandle – Pointer to the I3C master DMA driver handle.

typedef struct *_i3c_master_edma_handle* i3c_master_edma_handle_t

typedef struct *_i3c_master_edma_callback* i3c_master_edma_callback_t

i3c master callback functions.

struct __i3c_master_edma_callback

#include <fsl_i3c_edma.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)

Target asks for controller request.

void (*ibiCallback)(I3C_Type *base, *i3c_master_edma_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)

IBI event callback.

void (*transferComplete)(I3C_Type *base, *i3c_master_edma_handle_t* *handle, *status_t* status, void *userData)

Transfer complete callback.

struct __i3c_master_edma_handle

#include <fsl_i3c_edma.h> Driver handle for master EDMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

I3C_Type *base
: I3C base pointer.

uint8_t state
: Transfer state machine current state.

uint32_t transferCount
: Indicates progress of the transfer

uint8_t subaddressBuffer[4]
: Saving subaddress command.

uint8_t subaddressCount
: Saving command count.

i3c_master_transfer_t transfer
: Copy of the current transfer info.

i3c_master_edma_callback_t callback
: Callback function pointer.

void *userData
: Application data passed to callback.

edma_handle_t *rxDmaHandle
: Handle for receive DMA channel.

edma_handle_t *txDmaHandle
: Handle for transmit DMA channel.

bool ibiFlag
: IBIWON flag.

uint8_t ibiAddress
: Slave address which request IBI.

uint8_t *ibiBuff
: Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
: IBI payload size.

i3c_ibi_type_t ibiType
: IBI type.

status_t result
: Transfer result.

2.33 I3C Slave Driver

void I3C_SlaveGetDefaultConfig(*i3c_slave_config_t* *slaveConfig)

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave              = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

Parameters

- slaveConfig – **[out]** User provided configuration structure for default values. Refer to i3c_slave_config_t.

void I3C_SlaveInit(I3C_Type *base, const *i3c_slave_config_t* *slaveConfig, uint32_t slowClock_Hz)

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- slowClock_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH defines as 1, this parameter is useless.

void I3C_SlaveDeinit(I3C_Type *base)

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnable)

Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnable – Enable or disable.

static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_flags

Parameters

- base – The I3C peripheral base address.

Returns
State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Attempts to clear other flags has no effect.

See also:

_i3c_slave_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _i3c_slave_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_error_flags

Parameters

- base – The I3C peripheral base address.

Returns
State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave error status flag state.

See also:

_i3c_slave_error_flags.

Parameters

- base – The I3C peripheral base address.

- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of _i3c_slave_error_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)

Gets the I3C slave state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C slave activity state, refer i3c_slave_activity_state_t.

status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag

- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns
The I3C Slave Receive Data Register address.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The kI3C_SlaveTxReadyFlag flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The kI3C_SlaveRxReadyFlag flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns
Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns
Error or success status returned by API.

void I3C_SlaveTransferCreateHandle(I3C_Type *base, *i3c_slave_handle_t* *handle, *i3c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, *i3c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and I3C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to I3C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i3c_slave_transfer_event_t enumerators for the events you wish to receive. The kI3C_SlaveTransmitEvent and kI3C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI3C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to struct: _i3c_slave_handle structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together i3c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kI3C_SlaveAllEvents to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_I3C_Busy – Slave transfers have already been started on this handle.

status_t I3C_SlaveTransferGetCount(I3C_Type *base, *i3c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.

- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void I3C_SlaveTransferAbort(I3C_Type *base, *i3c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to struct: _i3c_slave_handle structure which stores the transfer state.

void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to struct: _i3c_slave_handle structure which stores the transfer state.

enum _i3c_slave_flags

I3C slave peripheral flags.

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag

- kI3C_SlaveEventSentFlag

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveNotStopFlag
Slave status not stop flag

enumerator kI3C_SlaveMessageFlag
Slave status message, indicating slave is listening to the bus traffic or responding

enumerator kI3C_SlaveRequiredReadFlag
Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator kI3C_SlaveRequiredWriteFlag
Slave status request write, master is doing SDR write to slave, except slave in ENTDAA mode

enumerator kI3C_SlaveBusDAAFlag
I3C bus is in ENTDAA mode

enumerator kI3C_SlaveBusHDRModeFlag
I3C bus is in HDR mode

enumerator kI3C_SlaveBusStartFlag
Start/Re-start event is seen since the bus was last cleared

enumerator kI3C_SlaveMatchedFlag
Slave address(dynamic/static) matched since last cleared

enumerator kI3C_SlaveBusStopFlag
Stop event is seen since the bus was last cleared

enumerator kI3C_SlaveRxReadyFlag
Rx data ready in rx buffer flag

enumerator kI3C_SlaveTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_SlaveDynamicAddrChangedFlag
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C_SlaveReceivedCCCFlag
Slave received Common command code

enumerator kI3C_SlaveErrorFlag
Error occurred flag

enumerator kI3C_SlaveHDRCommandMatchFlag
High data rate command match

enumerator kI3C_SlaveCCCHandledFlag
Slave received Common command code is handled by I3C module

enumerator kI3C_SlaveEventSentFlag
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C_SlaveIbiDisableFlag
Slave in band interrupt is disabled.

enumerator kI3C_SlaveMasterRequestDisabledFlag
Slave master request is disabled.

enumerator kI3C_SlaveHotJoinDisabledFlag
Slave Hot-Join is disabled.

enumerator kI3C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C_SlaveAllIrqFlags

enum _i3c_slave_error_flags
I3C slave error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveErrorOverrunFlag
Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C_SlaveErrorUnderrunFlag
Slave internal to-bus buffer/FIFO underrun

enumerator kI3C_SlaveErrorUnderrunNakFlag
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C_SlaveErrorTermFlag
Terminate error from master

enumerator kI3C_SlaveErrorInvalidStartFlag
Slave invalid start flag

enumerator kI3C_SlaveErrorSdrParityFlag
SDR parity error

enumerator kI3C_SlaveErrorHdrParityFlag
HDR parity error

enumerator kI3C_SlaveErrorHdrCRCFlag
HDR-DDR CRC error

enumerator kI3C_SlaveErrorS0S1Flag
S0 or S1 error

enumerator kI3C_SlaveErrorOverreadFlag
Over-read error

enumerator kI3C_SlaveErrorOverwriteFlag
Over-write error

enum _i3c_slave_event
I3C slave.event.

Values:

enumerator kI3C_SlaveEventNormal
Normal mode.

enumerator kI3C_SlaveEventIBI
In band interrupt event.

enumerator kI3C_SlaveEventMasterReq
Master request event.

enumerator kI3C_SlaveEventHotJoinReq
Hot-join event.

enum __i3c_slave_activity_state
I3C slave.activity state.

Values:

enumerator kI3C_SlaveNoLatency
Normal bus operation

enumerator kI3C_SlaveLatency1Ms
1ms of latency.

enumerator kI3C_SlaveLatency100Ms
100ms of latency.

enumerator kI3C_SlaveLatency10S
10s latency.

enum __i3c_slave_transfer_event
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I3C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kI3C_SlaveAddressMatchEvent
Received the slave address after a start or repeated start.

enumerator kI3C_SlaveTransmitEvent
Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI3C_SlaveReceiveEvent
Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveRequiredTransmitEvent
Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveStartEvent
A start/repeated start was detected.

enumerator kI3C_SlaveHDRCommandMatchEvent
Slave Match HDR Command.

enumerator kI3C_SlaveCompletionEvent
A stop was detected, completing the transfer.

enumerator kI3C_SlaveRequestSentEvent
Slave request event sent.

enumerator kI3C_SlaveReceivedCCCEvent
Slave received CCC event, need to handle by application.

enumerator kI3C_SlaveAllEvents
Bit mask of all available events.

typedef enum *_i3c_slave_event* i3c_slave_event_t
I3C slave.event.

typedef enum *_i3c_slave_activity_state* i3c_slave_activity_state_t
I3C slave.activity state.

typedef struct *_i3c_slave_config* i3c_slave_config_t
Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_i3c_slave_transfer_event* i3c_slave_transfer_event_t
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I3C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_i3c_slave_transfer* i3c_slave_transfer_t
I3C slave transfer structure.

typedef struct *_i3c_slave_handle* i3c_slave_handle_t

typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, *i3c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the I3C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)
Typedef for slave interrupt handler.

struct _i3c_slave_config
#include <fsl_i3c.h> Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave
: Whether to enable slave.

uint8_t staticAddr
: Static address.

uint16_t vendorID
: Device vendor ID(manufacture ID).

uint32_t partNumber
: Device part number info

uint8_t dcr
: Device characteristics register information.

uint8_t bcr
: Bus characteristics register information.

uint8_t hdrMode
: Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest
: Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error
: Whether to ignore S0/S1 error in SDR mode.

bool offline
: Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop
: Whether to assert start/stop status only the time slave is addressed.

uint32_t maxWriteLength
: Maximum write length.

uint32_t maxReadLength
: Maximum read length.

struct __i3c_slave_transfer
: *#include <fsl_i3c.h>* I3C slave transfer structure.

Public Members

uint32_t event
: Reason the callback is being invoked.

uint8_t *txData
: Transfer buffer

size_t txDataSize
: Transfer size

uint8_t *rxData
: Transfer buffer

size_t rxDataSize
: Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI3C_SlaveCompletionEvent.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct _i3c_slave_handle

#include <fsl_i3c.h> I3C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

i3c_slave_transfer_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

i3c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

size_t txFifoSize

Tx Fifo size

2.34 I3C Slave DMA Driver

void I3C_SlaveTransferCreateHandleEDMA(I3C_Type *base, *i3c_slave_edma_handle_t* *handle, *i3c_slave_edma_callback_t* callback, void *userData, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle)

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_SlaveTransferAbortDMA() API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.

- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.
- rxDmaHandle – Handle for the DMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

status_t I3C_SlaveTransferEDMA(I3C_Type *base, *i3c_slave_edma_handle_t* *handle, *i3c_slave_edma_transfer_t* *transfer, uint32_t eventMask)

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.
- transfer – The pointer to the transfer descriptor.
- eventMask – Bit mask formed by OR'ing together i3c_slave_transfer_event_t enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.
- kStatus_Fail – The transaction can't be set.

void I3C_SlaveTransferAbortEDMA(I3C_Type *base, *i3c_slave_edma_handle_t* *handle)

Abort a slave edma non-blocking transfer in a early time.

Parameters

- base – I3C peripheral base address
- handle – pointer to i3c_slave_edma_handle_t structure

void I3C_SlaveTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- i3cHandle – Pointer to the I3C slave DMA driver handle.

typedef struct *_i3c_slave_edma_handle* i3c_slave_edma_handle_t

typedef struct *_i3c_slave_edma_transfer* i3c_slave_edma_transfer_t

I3C slave transfer structure.

typedef void (*i3c_slave_edma_callback_t)(I3C_Type *base, *i3c_slave_edma_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

Param base
Base address for the I3C instance on which the event occurred.

Param handle
Pointer to slave DMA transfer handle.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct _i3c_slave_edma_transfer

#include <fsl_i3c_edma.h> I3C slave transfer structure.

Public Members

uint32_t event

Reason the callback is being invoked.

uint8_t *txData

Transfer buffer

size_t txDataSize

Transfer size

uint8_t *rxData

Transfer buffer

size_t rxDataSize

Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for kI3C_SlaveCompletionEvent.

struct _i3c_slave_edma_handle

#include <fsl_i3c_edma.h> I3C slave edma handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

I3C_Type *base

I3C base pointer.

i3c_slave_edma_transfer_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit
: Whether the last transfer was a transmit.

bool isDdrMode
: Whether this is HDR-DDR transfer.

uint32_t eventMask
: Mask of enabled events.

i3c_slave_edma_callback_t callback
: Callback function called at transfer event.

edma_handle_t *rxDmaHandle
: Handle for receive DMA channel.

edma_handle_t *txDmaHandle
: Handle for transmit DMA channel.

void *userData
: Callback parameter passed to callback.

2.35 INPUTMUX: Input Multiplexing Driver

enum _inputmux_connection_t
: INPUTMUX connections type.

Values:

enumerator kINPUTMUX_GpioPort8Pin0ToPintsel

enumerator kINPUTMUX_GpioPort8Pin1ToPintsel

enumerator kINPUTMUX_GpioPort8Pin2ToPintsel

enumerator kINPUTMUX_GpioPort8Pin3ToPintsel

enumerator kINPUTMUX_GpioPort8Pin4ToPintsel

enumerator kINPUTMUX_GpioPort8Pin5ToPintsel

enumerator kINPUTMUX_GpioPort8Pin6ToPintsel

enumerator kINPUTMUX_GpioPort8Pin7ToPintsel

enumerator kINPUTMUX_GpioPort8Pin8ToPintsel

enumerator kINPUTMUX_GpioPort8Pin9ToPintsel

enumerator kINPUTMUX_GpioPort8Pin10ToPintsel

enumerator kINPUTMUX_GpioPort8Pin11ToPintsel

enumerator kINPUTMUX_GpioPort8Pin12ToPintsel

enumerator kINPUTMUX_GpioPort8Pin13ToPintsel

enumerator kINPUTMUX_GpioPort8Pin14ToPintsel

enumerator kINPUTMUX_GpioPort8Pin15ToPintsel

enumerator kINPUTMUX_GpioPort8Pin16ToPintsel

enumerator kINPUTMUX_GpioPort8Pin17ToPintsel

enumerator kINPUTMUX_GpioPort8Pin18ToPintsel

enumerator kINPUTMUX_GpioPort8Pin19ToPintsel

enumerator kINPUTMUX_GpioPort8Pin20ToPintsel

enumerator kINPUTMUX_GpioPort8Pin21ToPintsel

enumerator kINPUTMUX_GpioPort8Pin22ToPintsel

enumerator kINPUTMUX_GpioPort8Pin23ToPintsel

enumerator kINPUTMUX_GpioPort8Pin24ToPintsel

enumerator kINPUTMUX_GpioPort8Pin25ToPintsel

enumerator kINPUTMUX_GpioPort8Pin26ToPintsel

enumerator kINPUTMUX_GpioPort8Pin27ToPintsel

enumerator kINPUTMUX_GpioPort8Pin28ToPintsel

enumerator kINPUTMUX_GpioPort8Pin29ToPintsel

enumerator kINPUTMUX_GpioPort8Pin30ToPintsel

enumerator kINPUTMUX_GpioPort8Pin31ToPintsel

enumerator kINPUTMUX_GpioPort10Pin0ToPintsel

enumerator kINPUTMUX_GpioPort10Pin1ToPintsel

enumerator kINPUTMUX_GpioPort10Pin2ToPintsel

enumerator kINPUTMUX_GpioPort10Pin3ToPintsel

enumerator kINPUTMUX_GpioPort10Pin4ToPintsel

enumerator kINPUTMUX_GpioPort10Pin5ToPintsel

enumerator kINPUTMUX_GpioPort10Pin6ToPintsel

enumerator kINPUTMUX_GpioPort10Pin7ToPintsel

enumerator kINPUTMUX_GpioPort10Pin8ToPintsel

enumerator kINPUTMUX_GpioPort10Pin9ToPintsel

enumerator kINPUTMUX_GpioPort10Pin10ToPintsel

enumerator kINPUTMUX_GpioPort10Pin11ToPintsel

enumerator kINPUTMUX_GpioPort10Pin12ToPintsel

enumerator kINPUTMUX_GpioPort10Pin13ToPintsel

enumerator kINPUTMUX_GpioPort10Pin14ToPintsel

enumerator kINPUTMUX_GpioPort10Pin15ToPintsel

enumerator kINPUTMUX_GpioPort10Pin16ToPintsel

enumerator kINPUTMUX_GpioPort10Pin17ToPintsel

DSP Interrupt.

enumerator kINPUTMUX_Flexcomm17ToDspInterrupt

enumerator kINPUTMUX_Flexcomm18ToDspInterrupt

enumerator kINPUTMUX_Flexcomm19ToDspInterrupt

enumerator kINPUTMUX_Flexcomm20ToDspInterrupt

enumerator kINPUTMUX_Pmc1ToDspInterrupt

enumerator kINPUTMUX_Gpio8Irq0ToDspInterrupt

enumerator kINPUTMUX_Gpio8Irq1ToDspInterrupt

enumerator kINPUTMUX_Gpio9Irq0ToDspInterrupt

enumerator kINPUTMUX_Gpio9Irq1ToDspInterrupt

enumerator kINPUTMUX_Gpio10Irq0ToDspInterrupt

enumerator kINPUTMUX_Gpio10Irq1ToDspInterrupt

enumerator kINPUTMUX_Wdt2ToDspInterrupt

enumerator kINPUTMUX_Wdt3ToDspInterrupt

enumerator kINPUTMUX_Mu0BToDspInterrupt

enumerator kINPUTMUX_Mu3BToDspInterrupt

enumerator kINPUTMUX_Utick1ToDspInterrupt

enumerator kINPUTMUX_Mrt1ToDspInterrupt

enumerator kINPUTMUX_OsEventTimerToDspInterrupt

enumerator kINPUTMUX_Ctimer5ToDspInterrupt

enumerator kINPUTMUX_Ctimer6ToDspInterrupt

enumerator kINPUTMUX_Ctimer7ToDspInterrupt

enumerator kINPUTMUX_Rtc1AlarmToDspInterrupt

enumerator kINPUTMUX_Rtc1WakeupToDspInterrupt

enumerator kINPUTMUX_I3c2ToDspInterrupt

enumerator kINPUTMUX_I3c3ToDspInterrupt

enumerator kINPUTMUX_MicfilToDspInterrupt

enumerator kINPUTMUX_HwvadToDspInterrupt

enumerator kINPUTMUX_Dma2Irq0ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq1ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq2ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq3ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq4ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq5ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq6ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq7ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq0ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq1ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq2ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq3ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq4ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq5ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq6ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq7ToDspInterrupt

enumerator kINPUTMUX_GpioInt0ToDspInterrupt

enumerator kINPUTMUX_GpioInt1ToDspInterrupt

enumerator kINPUTMUX_GpioInt2ToDspInterrupt

enumerator kINPUTMUX_GpioInt3ToDspInterrupt

enumerator kINPUTMUX_Sai3ToDspInterrupt

enumerator kINPUTMUX_CtInp0ToFlexcomm17

enumerator kINPUTMUX_CtInp1ToFlexcomm17

enumerator kINPUTMUX_CtInp2ToFlexcomm17

enumerator kINPUTMUX_CtInp3ToFlexcomm17

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm17

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm17

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm17

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm17

enumerator kINPUTMUX_Cmp0OutToFlexcomm17

enumerator kINPUTMUX_Rtc1IrqToFlexcomm17

enumerator kINPUTMUX_CtInp0ToFlexcomm18

enumerator kINPUTMUX_CtInp1ToFlexcomm18

enumerator kINPUTMUX_CtInp2ToFlexcomm18

enumerator kINPUTMUX_CtInp3ToFlexcomm18

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm18

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm18

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm18

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm18

enumerator kINPUTMUX_Cmp0OutToFlexcomm18

enumerator kINPUTMUX_Rtc1IrqToFlexcomm18

enumerator kINPUTMUX_CtInp0ToFlexcomm19

enumerator kINPUTMUX_CtInp1ToFlexcomm19

enumerator kINPUTMUX_CtInp2ToFlexcomm19

enumerator kINPUTMUX_CtInp3ToFlexcomm19

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm19

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm19

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm19

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm19

enumerator kINPUTMUX_Cmp0OutToFlexcomm19

enumerator kINPUTMUX_Rtc1IrqToFlexcomm19

enumerator kINPUTMUX_CtInp0ToFlexcomm20

enumerator kINPUTMUX_CtInp1ToFlexcomm20

enumerator kINPUTMUX_CtInp2ToFlexcomm20

enumerator kINPUTMUX_CtInp3ToFlexcomm20

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm20

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm20

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm20

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm20

enumerator kINPUTMUX_Cmp0OutToFlexcomm20

enumerator kINPUTMUX_Rtc1IrqToFlexcomm20

enumerator kINPUTMUX_Gpio0Irq0ToAdc0

enumerator kINPUTMUX_Gpio1Irq0ToAdc0

enumerator kINPUTMUX_Gpio2Irq0ToAdc0

enumerator kINPUTMUX_Gpio3Irq0ToAdc0

enumerator kINPUTMUX_Gpio4Irq0ToAdc0

enumerator kINPUTMUX_Gpio5Irq0ToAdc0

enumerator kINPUTMUX_Gpio6Irq0ToAdc0

enumerator kINPUTMUX_Gpio7Irq0ToAdc0

enumerator kINPUTMUX_Gpio8Irq0ToAdc0

enumerator kINPUTMUX_Gpio9Irq0ToAdc0

enumerator kINPUTMUX_Gpio10Irq0ToAdc0

enumerator kINPUTMUX_Sct0Out4ToAdc0

enumerator kINPUTMUX_Sct0Out5ToAdc0

enumerator kINPUTMUX_Sct0Out9ToAdc0

enumerator kINPUTMUX_Ct0Mat3ToAdc0

enumerator kINPUTMUX_Ct1Mat3ToAdc0

enumerator kINPUTMUX_Ct2Mat3ToAdc0

enumerator kINPUTMUX_Ct3Mat3ToAdc0

enumerator kINPUTMUX_Ct4Mat3ToAdc0

enumerator kINPUTMUX_Ct5Mat3ToAdc0

enumerator kINPUTMUX_Ct6Mat3ToAdc0

enumerator kINPUTMUX_Ct7Mat3ToAdc0

enumerator kINPUTMUX_Cmp0OutToAdc0

enumerator kINPUTMUX_Cpu0TxevToAdc0

enumerator kINPUTMUX_Cpu1TxevToAdc0

enumerator kINPUTMUX_EzhvOutToAdc0

enumerator kINPUTMUX_GpioInt0BmatchToAdc0

enumerator kINPUTMUX_GpioInt1BmatchToAdc0

CTmier5 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer5CaptureChannels

enumerator kINPUTMUX_Sai3TxSyncToTimer5CaptureChannels

enumerator kINPUTMUX_Sai3RxSyncToTimer5CaptureChannels

enumerator kINPUTMUX_Cmp0OutToTimer5CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp0ToTimer5CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp1ToTimer5CaptureChannels

CTmier6 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer6CaptureChannels

enumerator kINPUTMUX_Sai3TxSyncToTimer6CaptureChannels

enumerator kINPUTMUX_Sai3RxSyncToTimer6CaptureChannels

enumerator kINPUTMUX_Cmp0OutToTimer6CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp0ToTimer6CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp1ToTimer6CaptureChannels

CTmier7 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer7CaptureChannels

enumerator kINPUTMUX_Sai3TxSyncToTimer7CaptureChannels

enumerator kINPUTMUX_Sai3RxSyncToTimer7CaptureChannels

enumerator kINPUTMUX_Cmp0OutToTimer7CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp0ToTimer7CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp1ToTimer7CaptureChannels

CTmier5 input trigger.

enumerator kINPUTMUX_CtInp0ToTimer5Trigger

enumerator kINPUTMUX_CtInp1ToTimer5Trigger

enumerator kINPUTMUX_CtInp2ToTimer5Trigger

enumerator kINPUTMUX_CtInp3ToTimer5Trigger

enumerator kINPUTMUX_CtInp4ToTimer5Trigger

enumerator kINPUTMUX_CtInp5ToTimer5Trigger

enumerator kINPUTMUX_CtInp6ToTimer5Trigger

enumerator kINPUTMUX_CtInp7ToTimer5Trigger

enumerator kINPUTMUX_CtInp8ToTimer5Trigger

enumerator kINPUTMUX_CtInp9ToTimer5Trigger

enumerator kINPUTMUX_Sai3TxSyncToTimer5Trigger

enumerator kINPUTMUX_Sai3RxSyncToTimer5Trigger

enumerator kINPUTMUX_Cmp0OutToTimer5Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToTimer5Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToTimer5Trigger

CTmier6 input trigger.

enumerator kINPUTMUX_CtInp0ToTimer6Trigger

enumerator kINPUTMUX_CtInp1ToTimer6Trigger

enumerator kINPUTMUX_CtInp2ToTimer6Trigger

enumerator kINPUTMUX_CtInp3ToTimer6Trigger

enumerator kINPUTMUX_CtInp4ToTimer6Trigger

enumerator kINPUTMUX_CtInp5ToTimer6Trigger

enumerator kINPUTMUX_CtInp6ToTimer6Trigger

enumerator kINPUTMUX_CtInp7ToTimer6Trigger

enumerator kINPUTMUX_CtInp8ToTimer6Trigger

enumerator kINPUTMUX_CtInp9ToTimer6Trigger

enumerator kINPUTMUX_Sai3TxSyncToTimer6Trigger

enumerator kINPUTMUX_Sai3RxSyncToTimer6Trigger

enumerator kINPUTMUX_Cmp0OutToTimer6Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToTimer6Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToTimer6Trigger

CTmier7 input trigger.

enumerator kINPUTMUX_CtInp0ToTimer7Trigger

enumerator kINPUTMUX_CtInp1ToTimer7Trigger

enumerator kINPUTMUX_CtInp2ToTimer7Trigger

enumerator kINPUTMUX_CtInp3ToTimer7Trigger

enumerator kINPUTMUX_CtInp4ToTimer7Trigger

enumerator kINPUTMUX_CtInp5ToTimer7Trigger

enumerator kINPUTMUX_CtInp6ToTimer7Trigger

enumerator kINPUTMUX_CtInp7ToTimer7Trigger

enumerator kINPUTMUX_CtInp8ToTimer7Trigger

enumerator kINPUTMUX_CtInp9ToTimer7Trigger

enumerator kINPUTMUX_Sai3TxSyncToTimer7Trigger

enumerator kINPUTMUX_Sai3RxSyncToTimer7Trigger

enumerator kINPUTMUX_Cmp0OutToTimer7Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToTimer7Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToTimer7Trigger

typedef enum *_inputmux_connection_t* inputmux_connection_t
INPUTMUX connections type.

PINTSEL_PMUX_ID
Periphinmux IDs.

DSP_INT_PMUX_ID

PMUX_SHIFT

FSL_INPUTMUX_DRIVER_VERSION
Group interrupt driver version for SDK.

void INPUTMUX_Init(void *base)
Initialize INPUTMUX peripheral.

This function enables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

Return values
None. –

void INPUTMUX_AttachSignal(void *base, uint16_t index, *inputmux_connection_t* connection)
Attaches a signal.

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
```

In this example, INTMUX has 8 registers for PINT, PINT_SEL0~PINT_SEL7. With parameter index specified as 2, this function configures register PINT_SEL2.

Parameters

- base – Base address of the INPUTMUX peripheral.
- index – The serial number of destination register in the group of INPUTMUX registers with same name.
- connection – Applies signal from source signals collection to target signal.

Return values
None. –

void INPUTMUX_Deinit(void *base)

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

Return values
None. –

2.36 Iopctl_driver

typedef struct *_iopctl_group* iopctl_group_t

Array of IOPCTL pin definitions passed to IOPCTL_SetPinMuxing() must be in this format.

__STATIC_INLINE void IOPCTL_PinMuxSet (uint8_t port, uint8_t pin, uint32_t modefunc)

Sets I/O Pad Control pin mux.

Parameters

- port – : Port to mux
- pin – : Pin to mux
- modefunc – : OR'ed values of type IOPCTL_*

Returns
Nothing

__STATIC_INLINE void IOPCTL_SetPinMuxing (const iopctl_group_t *pinArray, uint32_t arrayLength)

Set all I/O Control pin muxing.

Parameters

- pinArray – : Pointer to array of pin mux selections
- arrayLength – : Number of entries in pinArray

Returns
Nothing

LPC_IOPCTL_DRIVER_VERSION

IOPCTL driver version 2.0.3.

IOPCTL_FUNC0

IOPCTL function and mode selection definitions.

Note: See the User Manual for specific modes and functions supported by the various pins.
Selects pin function 0

IOPCTL_FUNC1

Selects pin function 1

IOPCTL_FUNC2

Selects pin function 2

IOPCTL_FUNC3
: Selects pin function 3

IOPCTL_FUNC4
: Selects pin function 4

IOPCTL_FUNC5
: Selects pin function 5

IOPCTL_FUNC6
: Selects pin function 6

IOPCTL_FUNC7
: Selects pin function 7

IOPCTL_FUNC8
: Selects pin function 8

IOPCTL_FUNC9
: Selects pin function 9

IOPCTL_FUNC10
: Selects pin function 10

IOPCTL_FUNC11
: Selects pin function 11

IOPCTL_FUNC12
: Selects pin function 12

IOPCTL_FUNC13
: Selects pin function 13

IOPCTL_FUNC14
: Selects pin function 14

IOPCTL_FUNC15
: Selects pin function 15

IOPCTL_PUPD_EN
: Enables Pullup / Pulldown

IOPCTL_PULLDOWN_EN
: Selects pull-down function

IOPCTL_PULLUP_EN
: Selects pull-up function

IOPCTL_INBUF_EN
: Enables buffer function on input

IOPCTL_SLEW_RATE
: Slew Rate Control

IOPCTL_ANAMUX_EN
: Enables analog mux function by setting 0 to bit 7

IOPCTL_PSEDRAIN_EN
: Enables pseudo output drain function

IOPCTL_INV_EN
: Enables invert function on input

IOPCTL_DRIVE_100OHM
: Selects transmitter current drive 100ohm, avaiable for PIO0,1,3

IOPCTL_DRIVE_66OHM
: Selects transmitter current drive 66ohm, avaiable for PIO0,1,3

IOPCTL_DRIVE_50OHM
: Selects transmitter current drive 50ohm, avaiable for PIO0,1,3

IOPCTL_DRIVE_33OHM
: Selects transmitter current drive 33ohm, avaiable for PIO0,1,3

IOPCTL_FULLDRIVE_EN
: Selects full drive, avaiable for PIO2,8,9,10. (Unsupported for A0 Silicon)

uint32_t port

uint32_t pin

uint32_t modefunc

FSL_COMPONENT_ID

struct _iopctl_group
: *#include <fsl_iopctl.h>* Array of IOPCTL pin definitions passed to IOPCTL_SetPinMuxing() must be in this format.

2.37 IRTC: IRTC Driver

status_t IRTC_Init(RTC_Type *base, const *irtc_config_t* *config)

Ungates the IRTC clock and configures the peripheral for basic operation.

This function initiates a soft-reset of the IRTC module, this has not effect on DST, calendaring, standby time and tamper detect registers.

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

Parameters

- base – IRTC peripheral base address
- config – Pointer to user's IRTC config structure.

Returns
kStatus_Success If the driver is initialized successfully.

Returns
kStatus_Fail if we cannot disable register write protection

Returns
kStatus_InvalidArgument If the input parameters are wrong.

status_t IRTC_Deinit(RTC_Type *base)

Gate the IRTC clock.

Parameters

- base – IRTC peripheral base address

Returns
kStatus_Success If the driver is initialized successfully.

Returns
kStatus_InvalidArgument If the input parameters are wrong.

void IRTC_GetDefaultConfig(*irtc_config_t* *config)

Fill in the IRTC config struct with the default settings.

The default values are:

```
config->wakeupSelect = true;
config->timerStdMask = false;
config->alrmMatch = kRTC_MatchSecMinHr;
```

Parameters

- config – Pointer to user's IRTC config structure.

void IRTC_GetDatetime(RTC_Type *base, *irtc_datetime_t* *datetime)

Gets the IRTC time and stores it in the given time structure.

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to structure where the date and time details are stored.

status_t IRTC_SetAlarm(RTC_Type *base, const *irtc_datetime_t* *alarmTime)

Sets the IRTC alarm time.

Note: weekDay field of alarmTime is not used during alarm match and should be set to 0

Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to structure where the alarm time is stored.

Returns
kStatus_Success: success in setting the alarm kStatus_InvalidArgument: error in setting the alarm. Error occurs because the alarm datetime format is incorrect.

void IRTC_GetAlarm(RTC_Type *base, *irtc_datetime_t* *datetime)

Returns the IRTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the alarm date and time details are stored.

static inline void IRTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected IRTC interrupts.

Parameters

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

static inline void IRTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected IRTC interrupts.

Parameters

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

static inline uint32_t IRTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled IRTC interrupts.

Parameters

- base – IRTC peripheral base address

Returns

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

static inline uint32_t IRTC_GetStatusFlags(RTC_Type *base)

Gets the IRTC status flags.

Parameters

- base – IRTC peripheral base address

Returns

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

static inline void IRTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clears the IRTC status flags.

Parameters

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

void IRTC_GetDaylightTime(RTC_Type *base, *irtc_daylight_time_t* *datetime)

Gets the IRTC daylight savings time and stores it in the given time structure.

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to a structure where the date and time details are stored.

static inline void IRTC_EnableSubsecondCounter(RTC_Type *base, bool enable)

Enable the RTC wake-up timer.

1HZ clock out selected via call to API IRTC_ConfigClockOut in order for the subsecond counter to synchronize with the RTC_SECONDS counter.

Parameters

- base – RTC peripheral base address
- enable – Use/Un-use the sub-second counter.
  - true: Use RTC wake-up timer at the same time.
  - false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.

static inline uint32_t IRTC_GetSubsecondCount(RTC_Type *base)

Read the actual RTC sub-second COUNT value.

Parameters

- base – RTC peripheral base address

Returns
The actual RTC sub-second COUNT value.

static inline void IRTC_SetWakeupCount(RTC_Type *base, bool enable1kHzClk, uint32_t wakeupValue)

Set countdown value to the RTC wake timer counter register.

Parameters

- base – RTC peripheral base address
- enable1kHzClk – Enable 1kHz clock source for the wake timer, else use the 32kHz clock.
- wakeupValue – The value to be loaded into the WAKE register in wake timer counter.

static inline uint32_t IRTC_GetWakeupCount(RTC_Type *base)

Read the actual value from the WAKE register value in RTC wake timer.

Parameters

- base – RTC peripheral base address

Returns
The actual value of the WAKE register value in wake timer counter.

FSL_IRTC_DRIVER_VERSION

enum _irtc_clock_select

IRTC clock select.

Values:

enumerator kIRTC_Clk16K
16.384 kHz clock is selected.

enumerator kIRTC_Clk32K
32.768 kHz clock is selected.

enum _irtc_interrupt_enable

List of IRTC interrupts.

Values:

enumerator kIRTC_AlarmInterruptEnable
Alarm Interrupt Enable

enumerator kIRTC_DayInterruptEnable
Days Interrupt Enable

enumerator kIRTC_HourInterruptEnable
Hours Interrupt Enable

enumerator kIRTC_MinInterruptEnable
Minutes Interrupt Enable

enumerator kIRTC_1hzInterruptEnable
1 Hz interval Interrupt Enable

enumerator kIRTC_2hzInterruptEnable
2 Hz interval Interrupt Enable

enumerator kIRTC_4hzInterruptEnable
4 Hz interval Interrupt Enable

enumerator kIRTC_8hzInterruptEnable
8 Hz interval Interrupt Enable

enumerator kIRTC_16hzInterruptEnable
16 Hz interval Interrupt Enable

enumerator kIRTC_32hzInterruptEnable
32 Hz interval Interrupt Enable

enumerator kIRTC_64hzInterruptEnable
64 Hz interval Interrupt Enable

enumerator kIRTC_128hzInterruptEnable
128 Hz interval Interrupt Enable

enumerator kIRTC_256hzInterruptEnable
256 Hz interval Interrupt Enable

enumerator kIRTC_512hzInterruptEnable
512 Hz interval Interrupt Enable

enumerator kIRTC_WakeTimerInterruptEnable
Wake timer Interrupt Enable

enumerator kIRTC_TamperQueueFullInterruptEnable
Tamper queue full Interrupt Enable

enum _irtc_status_flags
List of IRTC flags.

Values:

enumerator kIRTC_AlarmFlag
Alarm Status flag

enumerator kIRTC_DayFlag
Days Status flag

enumerator kIRTC_HourFlag
Hour Status flag

enumerator kIRTC_MinFlag
Minutes Status flag

enumerator kIRTC_1hzFlag
1 Hz interval status flag

enumerator kIRTC_2hzFlag
2 Hz interval status flag

enumerator kIRTC_4hzFlag
4 Hz interval status flag

enumerator kIRTC_8hzFlag
8 Hz interval status flag

enumerator kIRTC_16hzFlag
16 Hz interval status flag

enumerator kIRTC_32hzFlag
32 Hz interval status flag

enumerator kIRTC_64hzFlag
64 Hz interval status flag

enumerator kIRTC_128hzFlag
128 Hz interval status flag

enumerator kIRTC_256hzFlag
256 Hz interval status flag

enumerator kIRTC_512hzFlag
512 Hz interval status flag

enumerator kIRTC_InvalidFlag
Indicates if time/date counters are invalid

enumerator kIRTC_WriteProtFlag
Write protect enable status flag

enumerator kIRTC_BusErrFlag
Bus error flag

enumerator kIRTC_WakeTimerFlag
Wake timer status flag

enum _irtc_alarm_match
IRTC alarm match options.

Values:

enumerator kRTC_MatchSecMinHr
Only match second, minute and hour

enumerator kRTC_MatchSecMinHrDay
Only match second, minute, hour and day

enumerator kRTC_MatchSecMinHrDayMnth
Only match second, minute, hour, day and month

enumerator kRTC_MatchSecMinHrDayMnthYr
Only match second, minute, hour, day, month and year

enum _irtc_clockout_sel
IRTC clockout select.

Values:

enumerator kIRTC_ClkoutNo
No clock out

enumerator kIRTC_ClkoutFine1Hz
clock out fine 1Hz

enumerator kIRTC_Clkout32kHz
clock out 32.768kHz

enumerator kIRTC_ClkoutCoarse1Hz
clock out coarse 1Hz

typedef enum *_irtc_clock_select* irtc_clock_select_t
IRTC clock select.

typedef enum *_irtc_interrupt_enable* irtc_interrupt_enable_t
List of IRTC interrupts.

typedef enum *_irtc_status_flags* irtc_status_flags_t

List of IRTC flags.

typedef enum *_irtc_alarm_match* irtc_alarm_match_t

IRTC alarm match options.

typedef enum *_irtc_clockout_sel* irtc_clockout_sel_t

IRTC clockout select.

typedef struct *_irtc_datetime* irtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_irtc_daylight_time* irtc_daylight_time_t

Structure is used to hold the daylight saving time.

typedef struct *_irtc_config* irtc_config_t

RTC config structure.

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

The config struct can be made const so it resides in flash

status_t IRTC_SetWriteProtection(RTC_Type *base, bool lock)

Locks or unlocks IRTC registers for write access.

Note: When the registers are unlocked, they remain in unlocked state for 2 seconds, after which they are locked automatically. After power-on-reset, the registers come out unlocked and they are locked automatically 15 seconds after power on.

Parameters

- base – IRTC peripheral base address
- lock – true: Lock IRTC registers; false: Unlock IRTC registers.

Returns

kStatus_Success: if lock or unlock operation is successful kStatus_Fail: if lock or unlock operation fails even after multiple retry attempts

static inline void IRTC_Reset(RTC_Type *base)

Performs a software reset on the IRTC module.

Clears contents of alarm, interrupt (status and enable except tamper interrupt enable bit) registers, STATUS[CMP_DONE] and STATUS[BUS_ERR]. This has no effect on DST, calendaring, standby time and tamper detect registers.

Parameters

- base – IRTC peripheral base address

void IRTC_ConfigClockOut(RTC_Type *base, *irtc_clockout_sel_t* clkOut)

Select which clock to output from RTC.

Select which clock to output from RTC for other modules to use inside SoC, for example, RTC subsystem needs RTC to output 1HZ clock for sub-second counter.

Parameters

- base – IRTC peripheral base address
- clkOut – select clock to use for output,

void IRTC_ConfigClockSelect(RTC_Type *base, *irtc_clock_select_t* clkSelect)

Select which clock is used by RTC.

Select which clock is used by RTC to output to the peripheral and divided to generate a 512 Hz clock and a 1 Hz clock.

Parameters

- base – IRTC peripheral base address
- clkSelect – select clock used by RTC

static inline void IRTC_EnableClockOutputToPeripheral(RTC_Type *base, bool enable)

Determines whether the selected clock is output to other peripherals.

Determines whether the selected clock is output to other peripherals.

Parameters

- base – IRTC peripheral base address
- enable – determine whether the selected clock is output to other peripherals

IRTC_STATUS_W1C_BITS

struct _irtc_datetime

#include <fsl_irtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1984 to 2239.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t weekDay

Range from 0(Sunday) to 6(Saturday).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _irtc_daylight_time

#include <fsl_irtc.h> Structure is used to hold the daylight saving time.

Public Members

uint8_t startMonth

Range from 1 to 12

uint8_t endMonth

Range from 1 to 12

uint8_t startDay
: Range from 1 to 31 (depending on month)

uint8_t endDay
: Range from 1 to 31 (depending on month)

uint8_t startHour
: Range from 0 to 23

uint8_t endHour
: Range from 0 to 23

struct _irtc_config
: *#include <fsl_irtc.h>* RTC config structure.

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

  The config struct can be made const so it resides in flash

  Public Members

  irtc_alarm_match_t alrmMatch
  : Pick one option from enumeration :: irtc_alarm_match_t

  irtc_clock_select_t clockSelect
  : Pick one option from enumeration :: irtc_clock_select_t

  bool disableClockOutput
  : true: The selected clock is not output to other peripherals; false: The selected clock is output to other peripherals

2.38 Intrusion and Tamper Response Controller

2.39 ITRC

status_t ITRC_SetActionToEvent(ITRC_Type *base, *itrc_out_signals_t* out, *itrc_input_signals_t* in, *itrc_lock_t* lock, *itrc_enable_t* enable)

Set ITRC Action to Event.

This function sets input Event signal to corresponding output Action response signal.

Parameters

- base – ITRC peripheral base address
- out – ITRC OUT signal action
- in – ITRC IN signal event
- lock – if set locks INx_SEL configuration. This can be cleared only by PMC Core reset.
- enable – if set input Event will be selected for output Action, otherwise disable (if not already locked).

Returns

kStatus_Success if success, kStatus_InvalidArgument otherwise

void ITRC_SetSWEvent0(ITRC_Type *base)

Trigger ITRC SW Event 0.

This funciton set SW_EVENT0 register with value !=0 which triggers ITRC SW Event 0.

Parameters

- base – ITRC peripheral base address

void ITRC_SetSWEvent1(ITRC_Type *base)

Trigger ITRC SW Event 1.

This funciton set SW_EVENT1 register with value !=0 which triggers ITRC SW Event 1.

Parameters

- base – ITRC peripheral base address

uint32_t ITRC_GetStatus(ITRC_Type *base)

Get ITRC Status.

This function returns ITRC register status.

Parameters

- base – ITRC peripheral base address

Returns

Value of ITRC STATUS register

status_t ITRC_ClearStatus(ITRC_Type *base, uint32_t word)

Clear ITRC status.

This function clears corresponding ITRC event or action in STATUS register.

Parameters

- base – ITRC peripheral base address
- word – 32bit word represent corresponding event/action in STATUS register to be cleared (see ITRC_STATUS_INx/OUTx_STATUS)

Returns

kStatus_Success if success, kStatus_InvalidArgument otherwise

status_t ITRC_ClearAllStatus(ITRC_Type *base)

Clear All ITRC status.

This function clears all event and action status.

Parameters

- base – ITRC peripheral base address

Returns

kStatus_Success if success

status_t ITRC_Init(ITRC_Type *base)

Initialize ITRC.

This function initializes ITRC by enabling IRQ.

Parameters

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

Returns

Status of the init operation

void ITRC_Deinit(ITRC_Type *base)

Deinitialize ITRC.

This function deinitializes ITRC by disabling IRQ.

Parameters

- base – ITRC peripheral base address

FSL_ITRC_DRIVER_VERSION

Defines ITRC driver version 2.4.0.

Change log:

- Version 2.4.0
  - Rework the input signal definition for better flexibility
- Version 2.3.0
  - Update names of kITRC_SwEvent1/2 to kITRC_SwEvent0/1 to align with RM
- Version 2.2.0
  - Update driver to new version and input events
- Version 2.1.0
  - Make SYSCON glitch platform dependent
- Version 2.0.0
  - initial version

enum _itrc_input_signals

Values:

enum _itrc_lock

Values:

enumerator kITRC_Unlock

enumerator kITRC_Lock

enum _itrc_enable

Values:

enumerator kITRC_Enable

enumerator kITRC_Disable

enum _itrc_out_signals

Values:

enumerator kITRC_Irq

enumerator kITRC_ElsReset

enumerator kITRC_PufZeroize

enumerator kITRC_RamZeroize

enumerator kITRC_ChipReset

enumerator kITRC_TamperOut

enumerator kITRC_TamperOut1

typedef enum *_itrc_input_signals* itrc_input_signals_t

typedef enum *_itrc_lock* itrc_lock_t

typedef enum *_itrc_enable* itrc_enable_t

typedef enum *_itrc_out_signals* itrc_out_signals_t

void ITRC0_DriverIRQHandler(void)

ITRC_STATUS_IN2_STATUS_MASK

ITRC_STATUS_IN3_STATUS_MASK

ITRC_STATUS_IN9_STATUS_MASK

ITRC_STATUS1_IN17_STATUS_MASK

ITRC_STATUS1_IN19_STATUS_MASK

ITRC_STATUS1_IN24_21_STATUS_MASK

ITRC_STATUS1_IN32_25_STATUS_MASK

ITRC_STATUS1_IN46_STATUS_MASK

IN_0_15_EVENTS_MASK

OUT_ACTIONS_MASK

ITRC_OUT_COUNT

ITRC

2.40 JPEGDEC: JPEG decoder Driver

Error codes for the JPEG decoder driver.

Values:

enumerator kStatus_JPEGDEC_NotSupported
Unsupported pixel format or image resolution.

enum _jpegdec_pixel_format
JPEG decoding pixel format.

Values:

enumerator kJPEGDEC_PixelFormatYUV420
YUV420 format, Y at first plane, UV at second plane.

enumerator kJPEGDEC_PixelFormatYUV422
YUV422, 1 planar in the YUYV sequence.

enumerator kJPEGDEC_PixelFormatRGB
RGB packed format.

enumerator kJPEGDEC_PixelFormatYUV444
YUV444, 1 planar in the YUVYUV sequence.

enumerator kJPEGDEC_PixelFormatGray
8-bit or 12-bit gray scale.

enumerator kJPEGDEC_PixelFormatReserved
Reserved.

enumerator kJPEGDEC_PixelFormatYCCK
YCCK format.

enum _jpegdec_pixel_depth
JPEG decoding pixel depth.

Values:

enumerator kJPEGDEC_PixelDepth8Bit
8-bit per pixel.

enumerator kJPEGDEC_PixelDepth12Bit
12-bit per pixel.

enum _jpegdec_flags
Decoder slot Interrupt status and enable/disable flags.

Values:

enumerator kJPEGDEC_StreamBufferHalfDoneFlag
Slot's bit stream buffer pointer has passed half the size.

enumerator kJPEGDEC_StreamBufferDoneFlag
Slot's bit stream buffer pointer has finished.

enumerator kJPEGDEC_SwitchInFlag
Current slot is switched in.

enumerator kJPEGDEC_DecodeCompleteFlag
Current slot finished decoding.

enumerator kJPEGDEC_DecodeErrorFlag
Current slot has error during decoding.

enumerator kJPEGDEC_DescptReadErrorFlag
Descriptor read error when switching in current slot.

enumerator kJPEGDEC_BitReadErrorFlag
Read JPEG stream error.

enumerator kJPEGDEC_PixelWriteErrorFlag
Write decoded pixel error.

enumerator kJPEGDEC_ErrorFlags

enumerator kJPEGDEC_AllFlags

enum _jpegdec_slots
JPEG decoding Slot.

Values:

enumerator kJPEGDEC_Slot0
Decoding slot 0.

enumerator kJPEGDEC_Slot1
Decoding slot 1.

enumerator kJPEGDEC_Slot2
Decoding slot 2.

enumerator kJPEGDEC_Slot3
Decoding slot 3.

enum _jpegdec_endian_mode
JPEG stream data format.

Values:

enumerator kJPEGDEC_BigEndian
JPEG data format is big endian.

enumerator kJPEGDEC_LittleEndian
JPEG data format is little endian.

typedef enum *_jpegdec_pixel_format* jpegdec_pixel_format_t
JPEG decoding pixel format.

typedef enum *_jpegdec_pixel_depth* jpegdec_pixel_depth_t
JPEG decoding pixel depth.

typedef struct *_jpegdec_decoder_config* jpegdec_decoder_config_t
JPEG decoder configuration.

typedef struct *_jpegdec_descpt* jpegdec_descpt_t
JPEG decoder descriptor.

This structure is used to configure the linked decoding descriptor for one or more slots in Single Bit Stream Repeat mode or Context Switch mode.

typedef enum *_jpegdec_endian_mode* jpegdec_endian_mode_t
JPEG stream data format.

typedef struct *_jpegdec_config* jpegdec_config_t
Configuration for JPEG decoder.

FSL_JPEGDEC_DRIVER_VERSION
JPEG decoder driver version.

void JPEGDEC_GetDefaultConfig(*jpegdec_config_t* *config)
Gets the default configuration for JPEG decoder.

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

Example:

```
config->endianMode = kJPEGDEC_LittleEndian;
config->slots      = 0xFU;
config->enable     = true;
config->enableLp   = false;
```

Parameters

- config – Pointer to JPEG decoder configuration structure.

void JPEGDEC_Init(*JPEG_DECODER_Type* *base, const *jpegdec_config_t* *config)
Initializes the JPEG decoder.

The default configuration can be got by calling JPEGDEC_GetDefaultConfig.

Parameters

- base – JPEG decoder peripheral base address.
- config – Pointer to JPEG decoder configuration structure.

void JPEGDEC_Deinit(*JPEG_DECODER_Type* *base)

Deinitializes the JPEG decoder.

Parameters

- base – JPEG decoder peripheral base address.

static inline void JPEGDEC_EnableInterrupts(*JPEG_DECODER_Type* *base, uint8_t slot, uint32_t mask)

Enables interrupts for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- mask – Interrupts mask formed by flags OR'ed together. _jpegdec_flags.

static inline void JPEGDEC_DisableInterrupts(*JPEG_DECODER_Type* *base, uint8_t slot, uint32_t mask)

Disables interrupts for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- mask – Interrupts mask formed by flags OR'ed together. _jpegdec_flags.

static inline uint32_t JPEGDEC_GetStatusFlags(*JPEG_DECODER_Type* *base, uint8_t slot)

Gets status flags for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.

Returns

Status flags asserted mask formed by flags OR'ed together. See "_jpegdec_flags".

static inline void JPEGDEC_ClearStatusFlags(*JPEG_DECODER_Type* *base, uint8_t slot, uint32_t mask)

Clears status flags for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- mask – Status flags mask formed by flags OR'ed together. See "_jpegdec_flags".

static inline void JPEGDEC_Enable(*JPEG_DECODER_Type* *base, bool enable)

Enables the JPEG decoder module.

Parameters

- base – JPEG decoder peripheral base address.
- enable – True to enable, false to disable.

static inline void JPEGDEC_EnableLp(*JPEG_DECODER_Type* *base, bool enable)

Enables the JPEG decoder core to enter low power mode.

Parameters

- base – JPEG decoder peripheral base address.
- enable – True to enable, false to disable.

static inline void JPEGDEC_Reset(*JPEG_DECODER_Type* *base)

Resets the JPEG decoder.

Parameters

- base – JPEG decoder peripheral base address.

static inline void JPEGDEC_StartDecode(*JPEG_DECODER_Type* *base)

Starts the JPEG decode.

After calling this API the JPEG decoder will start decoding on a frame basis. The decoding process stops after the frame decoding is complete and if there are no other valid descriptors available. Auto-clear.

Parameters

- base – JPEG decoder peripheral base address.

static inline void JPEGDEC_EnableSlots(*JPEG_DECODER_Type* *base, uint8_t slots)

Enables the JPEG decoding slot(s)

Parameters

- base – JPEG decoder peripheral base address.
- slot – Masks of the enabled slot, _jpegdec_slots.

status_t JPEGDEC_GetActiveSlot(*JPEG_DECODER_Type* *base, uint8_t *slot)

Gets the current active slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – **[out]** Pointer to the variable that stores the enabled slot.

Return values

- kStatus_Success – Successfully get the current active slot.
- kStatus_Fail – Current JPEG decoder is idle, no slot is active.

static inline void JPEGDEC_SetEndian(*JPEG_DECODER_Type* *base, *jpegdec_endian_mode_t* mode)

Sets the JPEG decoder endian mode for decoded data.

Parameters

- base – JPEG decoder peripheral base address.
- mode – Endian mode.

void JPEGDEC_SetJpegBuffer(*jpegdec_decoder_config_t* *config, uint8_t *buffer, size_t length)

Sets the address and length of the JPEG image.

Parameters

- config – **[out]** Decoder configuration.
- data – Start address of the buffer of the raw JPEG image, shall be 8-byte aligned.
- length – Size of the buffer.

void JPEGDEC_SetOutputBuffer(*jpegdec_decoder_config_t* *config, uint8_t *imageData0, uint8_t *imageData1)

Sets the address of the decoded data.

Parameters

- config – **[out]** Decoder configuration.
- imageData0 – Address of the output buffer0, shall be 16-byte aligned.
- imageData1 – Address of the output buffer1, only used when output format is YUV420, shall be 16-byte aligned.

status_t JPEGDEC_ParseHeader(*jpegdec_decoder_config_t* *config)

Parses the JPEG header and stores the info in the decoder configuration structure.

Parameters

- config – **[out]** Decoder configuration.

Return values

- kStatus_Success – Header parsing success.
- kStatus_Fail – JPEG header parsing failed due corrupted header.
- kStatus_JPEGDEC_NotSupported – Header parsing failed due to the image pixel format is not supported or image width/height is larger than 8k or image width/height is not 8-byte aligned.

void JPEGDEC_SetDecodeOption(*jpegdec_decoder_config_t* *config, uint16_t pitch, bool clearStreamBuf, bool autoStart)

Sets the decode option.

Parameters

- config – **[out]** Decoder configuration.
- pitch – Output buffer pitch.
- clearStreamBuf – Set to true to clear the saved bit stream buffer, not used in single frame mode.
- autoStart – Set to true to automatically start the decoding when this descriptor is switched on, not used in single frame mode.

void JPEGDEC_ConfigDecoder(*JPEG_DECODER_Type* *base, const *jpegdec_decoder_config_t* *config)

Sets JPEG decoder decoder configuration.

Call this API to set the decoder configuration and then call JPEGDEC_StartDecode to start decode. Need to enable at least one slot in jpegdec_config_t first.

Parameters

- base – JPEG decoder peripheral base address.
- config – Decoder configuration.

void JPEGDEC_SetSlotNextDescpt(*JPEG_DECODER_Type* *base, uint8_t slot, *jpegdec_descpt_t* *descriptor)

Sets the address of the next decoder descriptor for certain slot.

Call this API to set the decoder descriptor for certain slot, and then call JPEGDEC_EnableSlotNextDescpt to enable the descriptor. If the decode configuration of this descriptor has enabled autoStart then decode will start automatically. Otherwise user has to call JPEGDEC_StartDecode manually. Need to enable the slot in jpegdec_config_t first.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- descriptor – Pointer to the descriptor structure.

static inline void JPEGDEC_EnableSlotNextDescpt(*JPEG_DECODER_Type* *base, uint8_t slot)

Enables the next decoder descriptor for certain slot.

Call this API after calling JPEGDEC_SetSlotNextDescpt to set valid descriptor for certain slot. User can also call JPEGDEC_EnableLinkedDescpt before calling JPEGDEC_SetSlotNextDescpt, then the descriptor will be loaded automatically.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.

static inline void JPEGDEC_EnableLinkedDescpt(*jpegdec_descpt_t* *descriptor)

Enables the next linked descriptor.

Call this API before calling JPEGDEC_SetSlotNextDescpt, then the descriptor will be loaded automatically. Otherwise call JPEGDEC_EnableSlotNextDescpt after calling JPEGDEC_SetSlotNextDescpt.

Parameters

- descriptor – **[out]** Pointer to the descriptor structure.

void JPEGDEC_DescptReset(*jpegdec_descpt_t* *descriptor)

Sets all fields to default values for the descriptor structure.

Note: This function enables the auto start feature.

Parameters

- descriptor – Pointer to the descriptor structure.

JPEGDEC_Type *core

Pointer to decoder core.

JPGDECWRP_Type *wrapper

Pointer to decoder wrapper.

uint32_t outBufAddr0

Base address for output frame buffer0. 16-byte aligned(4 LSBs are zero).

uint32_t outBufAddr1

Base address for output frame buffer1, only used for YUV420 second plane. 16-byte aligned(4 LSBs are zero).

uint32_t outBufPitch

Output buffer pitch.

uint32_t ___pad0___

Reserved.

uint32_t jpegBufAddr

Base address of the JPEG stream buffer for decoding. 16-byte aligned(4 LSBs are

zero).

uint32_t jpegBufSize
: Size of the JPEG stream buffer. 1k-byte aligned(10 LSBs are zero).

uint32_t height
: Image height in pixel. Max supported is 8K(2000h), must be integer times of 8.

uint32_t ___pad1___
: Reserved.

uint32_t width
: Image width in pixel. Max supported is 8K(2000h), must be integer times of 8.

uint32_t ___pad2___
: Reserved.

uint32_t ___pad3___
: Reserved.

uint32_t pixelDepth
: Pixel depth, jpegdec_pixel_depth_t.

uint32_t pixelFormat
: The image pixel format, jpegdec_pixel_format_t.

uint32_t clearStreamBuf
: Set to true to clear the saved bit stream buffer.

uint32_t autoStart
: Set to true to automatically start the decoding when this descriptor is switched on.

uint32_t ___pad4___
: Reserved.

uint32_t nextDescptAddr
: Address of the next decoding descriptor.

jpegdec_decoder_config_t config
: The decoding configuration for this descriptor.

bool enable
: Enables the decode IP.

bool enableLp
: Enables the core to enter low power mode.

jpegdec_endian_mode_t endianMode
: Endian mode for the souce/decoded image.

uint8_t slots
: Mask of enabled slots _jpegdec_slots.

JPEG_ALIGN_SIZE(size, align)

struct JPEG_DECODER_Type
: *#include <fsl_jpegdec.h>* JPEG decoder structure definition.

struct _jpegdec_decoder_config
: *#include <fsl_jpegdec.h>* JPEG decoder configuration.

struct _jpegdec_descpt
: *#include <fsl_jpegdec.h>* JPEG decoder descriptor.

  This structure is used to configure the linked decoding descriptor for one or more slots in Single Bit Stream Repeat mode or Context Switch mode.

struct _jpegdec_config
#include <fsl_jpegdec.h> Configuration for JPEG decoder.

2.41 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

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*.

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.

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.

2.42 LCDIF: LCD interface

status_t LCDIF_Init(LCDIF_Type *base)
Initialize the LCDIF.

This function initializes the LCDIF to work.

Parameters

- base – LCDIF peripheral base address.

Return values
kStatus_Success – Initialize successfully.

void LCDIF_Deinit(LCDIF_Type *base)
De-initialize the LCDIF.

This function disables the LCDIF peripheral clock.

Parameters

- base – LCDIF peripheral base address.

void LCDIF_DpiModeGetDefaultConfig(*lcdif_dpi_config_t* *config)
Get the default configuration for to initialize the LCDIF.

The default configuration value is:

```
config->panelWidth = 0;
config->panelHeight = 0;
config->hsw = 0;
config->hfp = 0;
config->hbp = 0;
config->vsw = 0;
config->vfp = 0;
```

(continues on next page)

(continued from previous page)

```
config->vbp = 0;
config->polarityFlags = kLCDIF_VsyncActiveLow | kLCDIF_HsyncActiveLow | kLCDIF_
↪DataEnableActiveHigh |
kLCDIF_DriveDataOnFallingClkEdge; config->format = kLCDIF_Output24Bit;
```

Parameters

- config – Pointer to the LCDIF configuration.

status_t LCDIF_DpiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dpi_config_t* *config)

Initialize the LCDIF to work in DPI mode.

This function configures the LCDIF DPI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

status_t LCDIF_DbiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dbi_config_t* *config)

Initialize the LCDIF to work in DBI mode.

This function configures the LCDIF DBI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

void LCDIF_DbiModeGetDefaultConfig(*lcdif_dbi_config_t* *config)

Get the default configuration to initialize the LCDIF DBI mode.

The default configuration value is:

```
config->swizzle         = kLCDIF_DbiOutSwizzleRGB;
config->format          = kLCDIF_DbiOutD8RGB332;
config->acTimeUnit      = 0;
config->type            = kLCDIF_DbiTypeA_ClockedE;
config->reversePolarity = false;
config->writeWRPeriod   = 3U;
config->writeWRAssert   = 0U;
config->writeCSAssert   = 0U;
config->writeWRDeassert = 0U;
config->writeCSDeassert = 0U;
config->typeCTas        = 1U;
config->typeCSCLTwrl    = 1U;
config->typeCSCLTwrh    = 1U;
```

Parameters

- config – Pointer to the LCDIF DBI configuration.

static inline void LCDIF_DbiReset(LCDIF_Type *base, uint8_t displayIndex)

Reset the DBI module.

Parameters

- displayIndex – Display index.
- base – LCDIF peripheral base address.

static inline bool LCDIF_DbiIsTypeCFifoFull(LCDIF_Type *base, uint8_t displayIndex)

Check whether the FIFO is full in DBI mode type C.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

Return values

- true – FIFO full.
- false – FIFO not full.

void LCDIF_DbiSelectArea(LCDIF_Type *base, uint8_t displayIndex, uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY, bool isTiled)

Select the update area in DBI mode.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startX – X coordinate for start pixel.
- startY – Y coordinate for start pixel.
- endX – X coordinate for end pixel.
- endY – Y coordinate for end pixel.
- isTiled – true if the pixel data is tiled.

static inline void LCDIF_DbiSendCommand(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd)

Send command to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.

void LCDIF_DbiSendData(LCDIF_Type *base, uint8_t displayIndex, const uint8_t *data, uint32_t dataLen_Byte)

brief Send data to DBI port.

Can be used to send light weight data to panel. To send pixel data in frame buffer, use LCDIF_DbiWriteMem.

param base LCDIF peripheral base address. param displayIndex Display index. param data pointer to data buffer. param dataLen_Byte data buffer length in byte.

void LCDIF_DbiSendCommandAndData(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd, const uint8_t *data, uint32_t dataLen_Byte)

Send command followed by data to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.
- data – pointer to data buffer.
- dataLen_Byte – data buffer length in byte.

static inline void LCDIF_DbiWriteMem(LCDIF_Type *base, uint8_t displayIndex)

Send pixel data in frame buffer to panel controller memory.

This function starts sending the pixel data in frame buffer to panel controller, user can monitor interrupt kLCDIF_Display0FrameDoneInterrupt to know when then data sending finished.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

void LCDIF_SetFrameBufferConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_fb_config_t* *config)

Configure the LCDIF frame buffer.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the legacy video layer, and use LCDIF_SetOverlayFrameBufferConfig to configure the overlay layers.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

void LCDIF_FrameBufferGetDefaultConfig(*lcdif_fb_config_t* *config)

Get default frame buffer configuration.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to get the default configuration for all the 3 layers.

The default configuration is

```
config->enable = true;
config->enableGamma = false;
config->format = kLCDIF_PixelFormatRGB565;
```

Parameters

- config – Pointer to the configuration structure.

static inline void LCDIF_SetFrameBufferAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Set the frame buffer to LCDIF.

Note: The address must be 128 bytes aligned.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

void LCDIF_SetFrameBufferStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

Set the frame buffer stride.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- strideBytes – The stride in byte.

static inline void LCDIF_SetFrameBufferUVAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Set the frame buffer to LCDIF for UV plane when the input format is YUV420.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

static inline void LCDIF_SetFrameBufferUVStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

void LCDIF_SetFrameBufferPosition(LCDIF_Type *base, uint8_t displayIndex, uint16_t topLeftX, uint16_t topLeftY, uint16_t width, uint16_t height)

Configure the video layer position.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the legacy video layer, and use LCDIF_SetOverlayLayerPosition to configure the overlay layers.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- topLeftX – The x value of thr top-left coordinate.
- topLeftY – The y value of thr top-left coordinate.
- width – The width of the layer.
- height – The height of the layer.

void LCDIF_SetOverlayLayerConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_fb_config_t* *config, uint8_t layerIndex)

Configure the overlay layers for LCDIF frame buffer.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers

which shares the same configurations. Use this API to configure the overlay layers, and use LCDIF_SetFrameBufferConfig to configure the legacy video layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.
- layerIndex – Pointer to the configuration structure.

void LCDIF_SetOverlayLayerPosition(LCDIF_Type *base, uint8_t displayIndex, uint16_t topLeftX, uint16_t topLeftY, uint16_t width, uint16_t height, uint8_t layerIndex)

Configure the overlay layer position.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the overlay layers, and use LCDIF_SetFrameBufferPosition to configure the legacy video layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- topLeftX – The x value of thr top-left coordinate.
- topLeftY – The y value of thr top-left coordinate.
- width – The width of the layer.
- height – The height of the layer.
- layerIndex – Pointer to the configuration structure.

void LCDIF_SetOverlayLayerAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address, uint8_t layerIndex)

Sets the frame buffer address for overlay layer.

Note: The address must be 64 bytes aligned.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.
- layerIndex – Which layer to configure.

void LCDIF_SetOverlayLayerStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes, uint8_t layerIndex)

Sets the frame buffer stride for overlay layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- strideBytes – The stride in byte.
- layerIndex – Which layer to configure.

static inline void LCDIF_SetOverlayLayerUVAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Sets the frame buffer address for overlay layer 0 for UV plane when the input format is YUV420.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

static inline void LCDIF_SetOverlayLayerUVStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

static inline void LCDIF_SetFrameBufferBackground(LCDIF_Type *base, uint8_t displayIndex, uint32_t color)

Sets the color for background layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- color – Background color in ARGB8888 format.

status_t LCDIF_GetPorterDuffConfig(*lcdif_porter_duff_blend_mode_t* mode, *lcdif_layer_alpha_blend_config_t* *config)

brief Get the alpha blend configuration by porter duff blend mode.

param mode The blend mode. param config Pointer to the configuration. retval kStatus_Success Successfully get the configuration. retval kStatus_InvalidArgument The blend mode not supported.

void LCDIF_PanelGetDefaultConfig(*lcdif_panel_config_t* *config)

Gets default panel configuration.

The default configuration is

```
config->enable = true;
config->enableGamma = false;
config->order       = kLCDIF_VideoOverlay0Overlay1;
config->endian      = kLCDIF_NoSwap;
```

Parameters

- config – Pointer to the configuration structure.

void LCDIF_SetPanelConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_panel_config_t* *config)

Configure the LCDIF panel.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

static inline void LCDIF_SetLayerOrder(LCDIF_Type *base, uint8_t displayIndex, *lcdif_layer_order_t* order)

Sets the layer order for the color key and alpha blend precess.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- order – The order of the 3 configurable layers.

static inline void LCDIF_SetEndianMode(LCDIF_Type *base, uint8_t displayIndex, *lcdif_endian_mode_t* mode)

Sets the endian mode.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- mode – The byte order for the endian mode.

static inline void LCDIF_EnableUpdate(LCDIF_Type *base, bool enable)

Enables the module to update the new set of configuration in the next frame's VBLANK.

Parameters

- base – LCDIF peripheral base address.
- enable – True to enable, false to disable.

static inline void LCDIF_EnableGamma(LCDIF_Type *base, bool enable)

Enables the gamma correction.

Parameters

- base – LCDIF peripheral base address.
- enable – True to enable, false to disable.

static inline void LCDIF_EnablePanel(LCDIF_Type *base, bool enable)

Enables the panel output.

When disables, all pixels will be black which allows a panel to have correct timing without any pixels.

Parameters

- base – LCDIF peripheral base address.
- enable – True to enable, false to disable.

static inline void LCDIF_Start(LCDIF_Type *base)

Starts the interface mode transfer. Self-clear.

All registers will be copied to the working set and resets the pixel counter.

Parameters

- base – LCDIF peripheral base address.

static inline void LCDIF_Reset(LCDIF_Type *base)

Resets the module. Self-clear.

Parameters

- base – LCDIF peripheral base address.

static inline void LCDIF_SetUpdateReady(LCDIF_Type *base)

Set the update of the double buffer register ready.

Call this API after updating any double buffered register to let the change take effect in next v-blank.

Parameters

- base – LCDIF peripheral base address.

void LCDIF_SetDitherConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dither_config_t* *config)

Set the dither configuration.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Index to configure.
- config – Pointer to the configuration structure.

void LCDIF_SetGammaData(LCDIF_Type *base, uint8_t displayIndex, uint16_t startIndex, const uint32_t *gamma, uint16_t gammaLen)

Set the gamma translation values to the LCDIF gamma table.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startIndex – Start index in the gamma table that the value will be set to.
- gamma – The gamma values to set to the gamma table in LCDIF, could be defined using LCDIF_MAKE_GAMMA_VALUE.
- gammaLen – The length of the gamma.

static inline void LCDIF_EnableInterrupts(LCDIF_Type *base, uint32_t mask)

Enables LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to enable, pass in as OR'ed value of _lcdif_interrupt.

static inline void LCDIF_DisableInterrupts(LCDIF_Type *base, uint32_t mask)

Disable LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to disable, pass in as OR'ed value of _lcdif_interrupt.

static inline uint32_t LCDIF_GetAndClearInterruptPendingFlags(LCDIF_Type *base)

Get and clear LCDIF interrupt pending status.

Note: The interrupt must be enabled, otherwise the interrupt flags will not assert.

Parameters

- base – LCDIF peripheral base address.

Returns

The interrupt pending status.

void LCDIF_CursorGetDefaultConfig(*lcdif_cursor_config_t* *config)

Get the hardware cursor default configuration.

The default configuration values are:

```
config->enable = true;
config->format = kLCDIF_CursorMasked;
config->hotspotOffsetX = 0;
config->hotspotOffsetY = 0;
```

Parameters

- config – Pointer to the hardware cursor configuration structure.

void LCDIF_SetCursorConfig(LCDIF_Type *base, const *lcdif_cursor_config_t* *config)

Configure the cursor.

Parameters

- base – LCDIF peripheral base address.
- config – Cursor configuration.

static inline void LCDIF_SetCursorHotspotPosition(LCDIF_Type *base, uint16_t x, uint16_t y)

Set the cursor hotspot postion.

Parameters

- base – LCDIF peripheral base address.
- x – X coordinate of the hotspot, range 0 ~ 8191.
- y – Y coordinate of the hotspot, range 0 ~ 8191.

static inline void LCDIF_SetCursorBufferAddress(LCDIF_Type *base, uint32_t address)

Set the cursor memory address.

Parameters

- base – LCDIF peripheral base address.
- address – Memory address.

void LCDIF_SetCursorColor(LCDIF_Type *base, uint32_t background, uint32_t foreground)

Set the cursor color.

Parameters

- base – LCDIF peripheral base address.
- background – Background color, could be defined use LCDIF_MAKE_CURSOR_COLOR
- foreground – Foreground color, could be defined use LCDIF_MAKE_CURSOR_COLOR

FSL_LCDIF_DRIVER_VERSION

enum _lcdif_polarity_flags

LCDIF signal polarity flags.

Values:

enumerator kLCDIF_VsyncActiveLow

VSYNC active low.

enumerator kLCDIF_VsyncActiveHigh

VSYNC active high.

enumerator kLCDIF_HsyncActiveLow

HSYNC active low.

enumerator kLCDIF_HsyncActiveHigh
HSYNC active high.

enumerator kLCDIF_DataEnableActiveLow
Data enable line active low.

enumerator kLCDIF_DataEnableActiveHigh
Data enable line active high.

enumerator kLCDIF_DriveDataOnFallingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.

enumerator kLCDIF_DriveDataOnRisingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.

enum _lcdif_output_format
LCDIF DPI output format.

Values:

enumerator kLCDIF_Output16BitConfig1
16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator kLCDIF_Output16BitConfig2
16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator kLCDIF_Output16BitConfig3
16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator kLCDIF_Output18BitConfig1
18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator kLCDIF_Output18BitConfig2
18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator kLCDIF_Output24Bit
24-bit.

enum _lcdif_fb_format
LCDIF frame buffer pixel format.

Values:

enumerator kLCDIF_PixelFormatARGB4444
ARGB4444, 16-bit each pixel, 4-bit each element.

enumerator kLCDIF_PixelFormatARGB1555
ARGB1555, 16-bit each pixel, 5-bit each element.

enumerator kLCDIF_PixelFormatRGB565
RGB565, 16-bit each pixel.

enumerator kLCDIF_PixelFormatARGB8888
ARGB8888, 32-bit each pixel, 8-bit each element.

enumerator kLCDIF_PixelFormatRGB888
RGB888, 24-bit each pixel, 8-bit each element.

enumerator kLCDIF_PixelFormatARGB8565
ARGB8565, 24-bit each pixel.

enumerator kLCDIF_PixelFormatARGB8888Tiled
ARGB8888, 32-bit each pixel, 8-bit each element in tiled format, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatYUV422Tiled
YUV422, component order V-Y-U-Y, in tiled format, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatYUV420Tiled
YUV420, in tiled format, need 2 plane, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatRGB888Tiled
RGB888, 24-bit each pixel, 8-bit each element in tiled format, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatARGB8565Tiled
ARGB8565, 24-bit each pixel in tiled format, not supported in overlay layer 1.

enum _lcdif_layer_input_order
LCDIF layer input pixel order.

Values:

enumerator kLCDIF_PixelInputOrderARGB
Input order ARGB.

enumerator kLCDIF_PixelInputOrderRGBA
Input order RGBA.

enumerator kLCDIF_PixelInputOrderABGR
Input order ABGR.

enumerator kLCDIF_PixelInputOrderBGRA
Input order BGRA.

enum _lcdif_layer_rotate_flip
LCDIF layer rotation or flip configuration.

Values:

enumerator kLCDIF_Rotate0
Clock wise rotate 0 degree.

enumerator kLCDIF_Rotate90
Clock wise rotate 90 degree, not supported yet.

enumerator kLCDIF_Rotate180
Clock wise rotate 180 degree.

enumerator kLCDIF_Rotate270
Clock wise rotate 270 degree, not supported yet.

enumerator kLCDIF_FlipHorizontal
Horizontal flip.

enumerator kLCDIF_FlipVertical
Vertical flip.

enum _lcdif_layer_decompress_mode
LCDIF layer input data decompress mode.

Values:

enumerator kLCDIF_DecompressNone
Decompression disabled. The input is straight RGB/YUV data.

enumerator kLCDIF_DecompressDECNanoNoneSample
DECNano decompression with none sample.

enumerator kLCDIF_DecompressDECNanoHSample
DECNano decompression with horizontal sample.

enumerator kLCDIF_DecompressDECNanoHVSample
DECNano decompression with both horizontal and vertical sample.

enumerator kLCDIF_DecompressETC2
ETC2 decompression.

Alpha blend alpha component mode. .

Values:

enumerator kLCDIF_AlphaStraight
Use straight alpha, s0_alpha' = s0_alpha.

enumerator kLCDIF_AlphaInversed
Use inversed alpha, s0_alpha' = 0xFF - s0_alpha.

LCDIF layer global alpha mode for alpha blend. .

Values:

enumerator kLCDIF_AlphaLocal
Use the alpha(or inversed alpha) component directly.

enumerator kLCDIF_AlphaGlobal
Use the global alpha instead.

enumerator kLCDIF_AlphaScaled
Multiply the alpha(or inversed alpha) component by the global alpha.

Alpha blend factor mode. .

Values:

enumerator kLCDIF_AlphaFactorZero
Use 0 as the blending factor.

enumerator kLCDIF_AlphaFactorOne
Use 1 as the blending factor.

enumerator kLCDIF_AlphaFactorStraight
Use the modified alpha component(As'' for source layer or Ad'' for destination layer) as the blending factor.

enumerator kLCDIF_AlphaFactorInversed
Use the inversed alpha component(1-As'' for source layer or 1-Ad'' for destination layer) as the blending factor.

enum _lcdif_porter_duff_blend_mode
LCDIF Porter Duff blend mode. Note: don't change the enum item value.

Values:

enumerator kLCDIF_PorterDuffSrc
Source Only

enumerator kLCDIF_PorterDuffAtop
Source Atop

enumerator kLCDIF_PorterDuffOver
Source Over

enumerator kLCDIF_PorterDuffIn
Source In.

enumerator kLCDIF_PorterDuffOut
Source Out.

enumerator kLCDIF_PorterDuffDst
Destination Only.

enumerator kLCDIF_PorterDuffDstAtop
Destination Atop.

enumerator kLCDIF_PorterDuffDstOver
Destination Over.

enumerator kLCDIF_PorterDuffDstIn
Destination In.

enumerator kLCDIF_PorterDuffDstOut
Destination Out.

enumerator kLCDIF_PorterDuffPlus
Clear.

enumerator kLCDIF_PorterDuffXor
XOR.

enumerator kLCDIF_PorterDuffClear
Clear.

enumerator kLCDIF_PorterDuffMax

enum _lcdif_layer_convert_standard
LCDIF Color space convert standard between RGB and YUV when the input format is tiled YUV.

Values:

enumerator kLCDIF_ConvertBT601
Use standard BT601.

enumerator kLCDIF_ConvertBT709
Use standard BT709.

enum _lcdif_layer_order
LCDIF the layer order from bottom to top.

There are 5 layers in total, apart from the background layer fixed in the bottom, and the cursor layer on top that does not participate in the color key and alpha blend, the order of the other 3 layers video/graphic layer, overlay layer 0 and 1 can be changed. During color key or alpha blend process, between 2 adjacent layers, the lower layer is considered as destination layer and the upper layer is considered as source layer. The precess is performed from bottom to top, after processing 2 layes, the generated new layer will be considered as destination layer then participate in the next process with the next upper layer.

Values:

enumerator kLCDIF_VideoOverlay0Overlay1
In the order of video, overlay0, overlay1.

enumerator kLCDIF_VideoOverlay1Overlay0
In the order of video, overlay1, overlay0.

enumerator kLCDIF_Overlay0VideoOverlay1
In the order of overlay0, video, overlay1.

enumerator kLCDIF_Overlay0Overlay1Video
In the order of overlay0, overlay1, video.

enumerator kLCDIF_Overlay1VideoOverlay0
In the order of overlay1, video, overlay0.

enumerator kLCDIF_Overlay1Overlay0Video
In the order of overlay1, overlay0, video.

enum _lcdif_endian_mode
LCDIF the pixel source endian mode.

Values:

enumerator kLCDIF_NoSwap
No swap.

enumerator kLCDIF_HalfWordSwap
16-bit boundary swap, from 0123 to 1032.

enumerator kLCDIF_WordSwap
32-bit boundary swap, from 0123 to 3210.

enum _lcdif_interrupt
LCDIF interrupt and status.

Values:

enumerator kLCDIF_Display0FrameDoneInterrupt
The last pixel of visible area in frame is shown.

enumerator kLCDIF_Display0DbiConfigErrorInterrupt
The DBI command is illegal.

enumerator kLCDIF_PanelUnderflowInterrupt
The AXI clock and pixel clock mismatch causing data supply insufficient.

enumerator kLCDIF_SoftwareResetDoneInterrupt
A software reset has completed.

enumerator kLCDIF_BusErrorInterrupt
An AXI bus transfer error occurrs.

enum _lcdif_cursor_format
LCDIF cursor format.

Values:

enumerator kLCDIF_CursorMasked
Masked format.

enumerator kLCDIF_CursorARGB8888
ARGB8888.

enum _lcdif_dbi_cmd_flag
LCDIF DBI command flag.

Values:

enumerator kLCDIF_DbiCmdAddress
Send address (or command).

enumerator kLCDIF_DbiCmdWriteMem
Start write memory.

enumerator kLCDIF_DbiCmdData
Send data.

enumerator kLCDIF_DbiCmdReadMem
Start read memory.

enum _lcdif_dbi_out_format
LCDIF DBI output format.

Values:

enumerator kLCDIF_DbiOutD8RGB332
8-bit data bus width, pixel RGB332. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD8RGB444
8-bit data bus width, pixel RGB444. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD8RGB565
8-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD8RGB666
8-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 3 cycles, data bus 2 LSB not used.

enumerator kLCDIF_DbiOutD8RGB888
8-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 3 cycles.

enumerator kLCDIF_DbiOutD9RGB666
9-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB332
16-bit data bus width, pixel RGB332. For type A or B. 2 pixels sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB444
16-bit data bus width, pixel RGB444. For type A or B. 1 pixel sent in 1 cycle, data bus 4 MSB not used.

enumerator kLCDIF_DbiOutD16RGB565
16-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB666Option1
16-bit data bus width, pixel RGB666. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB666Option2
16-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option1
16-bit data bus width, pixel RGB888. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option2
16-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD1RGB565Option1
1-bit data bus width, pixel RGB565. For type C option 1, use extra bit to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RGB565Option2

1-bit data bus width, pixel RGB565. For type C option 2, use extra byte to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RGB565Option3

1-bit data bus width, pixel RGB565. For type C option 3, use extra DC line to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RG888Option1

1-bit data bus width, pixel RGB888. For type C option 1, use extra bit to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RG888Option2

1-bit data bus width, pixel RGB888. For type C option 2, use extra byte to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RG888Option3

1-bit data bus width, pixel RGB888. For type C option 3, use extra DC line to distinguish Data and Command (DC).

enum _lcdif_dbi_type

LCDIF DBI type.

Values:

enumerator kLCDIF_DbiTypeA_FixedE

Selects DBI type A fixed E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeA_ClockedE

Selects DBI Type A Clocked E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeB

Selects DBI type B, 8080, Intel mode.

enumerator kLCDIF_DbiTypeC

Selects DBI type C, SPI mode.

enum _lcdif_dbi_out_swizzle

LCDIF DBI output swizzle.

Values:

enumerator kLCDIF_DbiOutSwizzleRGB

RGB

enumerator kLCDIF_DbiOutSwizzleBGR

BGR

typedef enum *_lcdif_output_format* lcdif_output_format_t

LCDIF DPI output format.

typedef struct *_lcdif_dpi_config* lcdif_dpi_config_t

Configuration for LCDIF module to work in DBI mode.

typedef enum *_lcdif_fb_format* lcdif_fb_format_t

LCDIF frame buffer pixel format.

typedef enum *_lcdif_layer_input_order* lcdif_layer_input_order_t

LCDIF layer input pixel order.

typedef enum *_lcdif_layer_rotate_flip* lcdif_layer_rotate_flip_t

LCDIF layer rotation or flip configuration.

typedef struct *_lcdif_layer_colorkey* lcdif_layer_colorkey_t

LCDIF layer color key configuration.

typedef enum *_lcdif_layer_decompress_mode* lcdif_layer_decompress_mode_t

LCDIF layer input data decompress mode.

typedef enum *_lcdif_porter_duff_blend_mode* lcdif_porter_duff_blend_mode_t

LCDIF Porter Duff blend mode. Note: don't change the enum item value.

typedef enum *_lcdif_layer_convert_standard* lcdif_layer_convert_standard_t

LCDIF Color space convert standard between RGB and YUV when the input format is tiled YUV.

typedef struct *_lcdif_fb_config* lcdif_fb_config_t

LCDIF frame buffer configuration.

For LCDIF of DC8000 version, there are 3 layers in the pre-processing, all shares the same frame buffer configuration.

typedef enum *_lcdif_layer_order* lcdif_layer_order_t

LCDIF the layer order from bottom to top.

There are 5 layers in total, apart from the background layer fixed in the bottom, and the cursor layer on top that does not participate in the color key and alpha blend, the order of the other 3 layers video/graphic layer, overlay layer 0 and 1 can be changed. During color key or alpha blend process, between 2 adjacent layers, the lower layer is considered as destination layer and the upper layer is considered as source layer. The precess is performed from bottom to top, after processing 2 layes, the generated new layer will be considered as destination layer then participate in the next process with the next upper layer.

typedef enum *_lcdif_endian_mode* lcdif_endian_mode_t

LCDIF the pixel source endian mode.

typedef struct *_lcdif_panel_config* lcdif_panel_config_t

LCDIF panel configuration.

typedef enum *_lcdif_cursor_format* lcdif_cursor_format_t

LCDIF cursor format.

typedef struct *_lcdif_cursor_config* lcdif_cursor_config_t

LCDIF cursor configuration.

typedef struct *_lcdif_dither_config* lcdif_dither_config_t

LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

typedef enum *_lcdif_dbi_out_format* lcdif_dbi_out_format_t

LCDIF DBI output format.

typedef enum *_lcdif_dbi_type* lcdif_dbi_type_t
LCDIF DBI type.

typedef enum *_lcdif_dbi_out_swizzle* lcdif_dbi_out_swizzle_t
LCDIF DBI output swizzle.

typedef struct *_lcdif_dbi_config* lcdif_dbi_config_t
LCDIF DBI configuration.

LCDIF_MAKE_CURSOR_COLOR(r, g, b)
Construct the cursor color, every element should be in the range of 0 ~ 255.

LCDIF_MAKE_GAMMA_VALUE(r, g, b)
Construct the gamma value set to LCDIF gamma table, every element should be in the range of 0~255.

LCDIF_MAKE_BACKGROUND_VALUE(a, r, g, b)
Construct the colour value for the background layer.

LCDIF_ALIGN_ADDR(addr, align)
Calculate the aligned address for LCDIF buffer.

LCDIF_FB_ALIGN
The frame buffer should be 128 byte aligned.

LCDIF_FB_UV_ALIGN

LCDIF_GAMMA_INDEX_MAX
Gamma index max value.

LCDIF_CURSOR_SIZE
The cursor size is 32 x 32.

LCDIF_FRAMEBUFFERCONFIG0_OUTPUT_MASK

LCDIF_ADDR_CPU_2_IP(addr)

struct lcdif_dpi_config
#include <fsl_lcdif.h> Configuration for LCDIF module to work in DBI mode.

Public Members

uint16_t panelWidth
Display panel width, pixels per line.

uint16_t panelHeight
Display panel height, how many lines per panel.

uint8_t hsw
HSYNC pulse width.

uint8_t hfp
Horizontal front porch.

uint8_t hbp
Horizontal back porch.

uint8_t vsw
VSYNC pulse width.

uint8_t vfp
Vrtical front porch.

uint8_t vbp
Vertical back porch.

uint32_t polarityFlags
OR'ed value of _lcdif_polarity_flags, used to contol the signal polarity.

lcdif_output_format_t format
DPI output format.

struct __lcdif_layer_colorkey
#include <fsl_lcdif.h> LCDIF layer color key configuration.

Public Members

bool enable
Enable the color keying.

struct lcdif_layer_alpha_blend_config_t
#include <fsl_lcdif.h> LCDIF frame buffer alpha blend configuration.

The alpha blending formula is C = Fs x Cs + Fd x Cd, and A = Fs x As'' + Fd x Ad''. C/A: generated final color/alpha component, Cs/Cd: original color component for source and destination layer, As''/Ad'': modified alpha component for source and destination layer, take As'' as example, As' = As or (1 - As), chosen by srcAlphaMode As'' = As' or Ags or (As' x Ags), chosen by srcGlobalAlphaMode, Ags configured by srcGlobalAlpha Fs/Fd: blending factor for source/destination layer, take Fs as example, Fs = 0 or 1 or As'' or (1 - As''), chosen by srcFactorMode

Public Members

uint32_t enable
Enables or disables alpha blend configuration.

uint32_t srcAlphaMode
Use the straight(As) or inversed(1-As) alpha value for modified alpha for the source layer, see lcdif_layer_alpha_mode_t.

uint32_t srcGlobalAlphaMode
Source layer global alpha mode, see lcdif_layer_global_alpha_mode_t.

uint32_t srcFactorMode
Source layer blend factor mode, see lcdif_layer_alpha_factor_mode_t.

uint32_t useSrcAlpha
Set to true to use source alpha for the source blending factor, otherwise use the destination alpha.

uint32_t dstAlphaMode
Use the straight(Ad) or inversed(1-Ad) alpha value for modified alpha for the destination layer, see lcdif_layer_alpha_mode_t.

uint32_t dstGlobalAlphaMode
Destination layer global alpha mode, see lcdif_layer_global_alpha_mode_t.

uint32_t dstFactorMode
Destination layer blend factor mode, see lcdif_layer_alpha_factor_mode_t.

uint32_t useDstAlpha
Set to true to use destination alpha for the destination blending factor, otherwise use the source alpha.

uint32_t srcGlobalAlpha

Source layer global alpha value, 0~255.

uint32_t dstGlobalAlpha

Destination layer global alpha value, 0~255.

struct __lcdif_fb_config

#include <fsl_lcdif.h> LCDIF frame buffer configuration.

For LCDIF of DC8000 version, there are 3 layers in the pre-processing, all shares the same frame buffer configuration.

Public Members

bool enable

Enable the layer output.

bool enableClear

Enable the color clear for this layer, onve enabled, the whole layer will be filled with the color configured in clearValue.

uint32_t clearValue

The value used for layer clear.

lcdif_fb_format_t format

Frame buffer output pixel format.

lcdif_layer_convert_standard_t standard

Color space convert standard, not used for overlay layer 1 since it does not support tile YUV input.

bool enableUVSwizzle

Swizzle the U/V components, not used for overlay layer 1 since it does not support tile YUV input.

lcdif_layer_input_order_t inOrder

Color component order of the input pixel.

lcdif_layer_colorkey_t colorkey

Color key configuration.

lcdif_layer_decompress_mode_t decompress

LCDIF layer input data decompress mode.

lcdif_layer_rotate_flip_t rotateFlipMode

LCDIF frame buffer rotation or flip configuration.

lcdif_layer_alpha_blend_config_t alpha

The alpha blending configuration.

uint16_t topLeftX

The x value of thr top-left coordinate.

uint16_t topLeftY

The y value of thr top-left coordinate.

uint16_t width

The width of the layer.

uint16_t height

The height of the layer.

struct __lcdif_panel_config

#include <fsl_lcdif.h> LCDIF panel configuration.

Public Members

bool enable

Enable the frame buffer output.

bool enableGamma

Enable the gamma correction.

lcdif_layer_order_t order

Layer order.

lcdif_endian_mode_t endian

Pixel source endian mode.

struct __lcdif_cursor_config

#include <fsl_lcdif.h> LCDIF cursor configuration.

Public Members

bool enable

Enable the cursor or not.

lcdif_cursor_format_t format

Cursor format.

uint8_t hotspotOffsetX

Offset of the hotspot to top left point, range 0 ~ 31

uint8_t hotspotOffsetY

Offset of the hotspot to top left point, range 0 ~ 31

struct __lcdif_dither_config

#include <fsl_lcdif.h> LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

Public Members

bool enable

Enable or not.

uint32_t low

Low part of the look up table.

uint32_t high
: High part of the look up table.

struct __lcdif_dbi_config
: *#include <fsl_lcdif.h>* LCDIF DBI configuration.

Public Members

lcdif_dbi_out_swizzle_t swizzle
: Swizzle.

lcdif_dbi_out_format_t format
: Output format.

uint8_t acTimeUnit
: Time unit for AC characteristics.

lcdif_dbi_type_t type
: DBI type.

bool reversePolarity
: Reverse the DC pin polarity.

uint16_t writeWRPeriod
: WR signal period, Cycle number = writeWRPeriod * (acTimeUnit + 1), must be no less than 3. Only for type A and type b.

uint8_t writeWRAssert
: Cycle number = writeWRAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to assert E. With kLCDIF_DbiTypeB: Time to assert WRX.

uint8_t writeCSAssert
: Cycle number = writeCSAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to assert CSX.

uint16_t writeWRDeassert
: Cycle number = writeWRDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to de-assert E. With kLCDIF_DbiTypeB: Time to de-assert WRX.

uint16_t writeCSDeassert
: Cycle number = writeCSDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to de-assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to de-assert CSX.

uint8_t typeCTas
: How many sdaClk cycles in Tas phase, only for Type C option 3, at least 1.

uint8_t typeCSCLTwrl
: How many sdaClk cycles in Twrl phase, only for Type C, at least 1.

uint8_t typeCSCLTwrh
: How many sdaClk cycles in Twrh phase, only for Type C, at least 1.

2.43 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum _lpadc_status_flags

Define hardware flags of the module.

Values:

enumerator kLPADC_ResultFIFO0OverflowFlag

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC_ResultFIFO0ReadyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC_TriggerExceptionFlag

Indicates that a trigger exception event has occurred.

enumerator kLPADC_TriggerCompletionFlag

Indicates that a trigger completion event has occurred.

enumerator kLPADC_CalibrationReadyFlag

Indicates that the calibration process is done.

enumerator kLPADC_ActiveFlag

Indicates that the ADC is in active state.

enumerator kLPADC_ResultFIFOOverflowFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowFlag as instead.

enumerator kLPADC_ResultFIFOReadyFlag

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0ReadyFlag as instead.

enum _lpadc_interrupt_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_ResultFIFO0OverflowInterruptEnable as instead.

enumerator kLPADC_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use kLPADC_FIFO0WatermarkInterruptEnable as instead.

enumerator kLPADC_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enumerator kLPADC_Trigger2CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 2 completion.

enumerator kLPADC_Trigger3CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 3 completion.

enumerator kLPADC_Trigger4CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 4 completion.

enumerator kLPADC_Trigger5CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 5 completion.

enumerator kLPADC_Trigger6CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 6 completion.

enumerator kLPADC_Trigger7CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 7 completion.

enumerator kLPADC_Trigger8CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 8 completion.

enumerator kLPADC_Trigger9CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 9 completion.

enumerator kLPADC_Trigger10CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 10 completion.

enumerator kLPADC_Trigger11CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 11 completion.

enumerator kLPADC_Trigger12CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 12 completion.

enumerator kLPADC_Trigger13CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 13 completion.

enumerator kLPADC_Trigger14CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 14 completion.

enumerator kLPADC_Trigger15CompletionInterruptEnable
Configures ADC to generate interrupt when trigger 15 completion.

enum _lpadc_trigger_status_flags
The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_Trigger0InterruptedFlag
Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC_Trigger1InterruptedFlag
Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC_Trigger2InterruptedFlag
Trigger 2 is interrupted by a high priority exception.

enumerator kLPADC_Trigger3InterruptedFlag
Trigger 3 is interrupted by a high priority exception.

enumerator kLPADC_Trigger4InterruptedFlag
Trigger 4 is interrupted by a high priority exception.

enumerator kLPADC_Trigger5InterruptedFlag
Trigger 5 is interrupted by a high priority exception.

enumerator kLPADC_Trigger6InterruptedFlag
Trigger 6 is interrupted by a high priority exception.

enumerator kLPADC_Trigger7InterruptedFlag
Trigger 7 is interrupted by a high priority exception.

enumerator kLPADC_Trigger8InterruptedFlag
Trigger 8 is interrupted by a high priority exception.

enumerator kLPADC_Trigger9InterruptedFlag
Trigger 9 is interrupted by a high priority exception.

enumerator kLPADC_Trigger10InterruptedFlag
Trigger 10 is interrupted by a high priority exception.

enumerator kLPADC_Trigger11InterruptedFlag
Trigger 11 is interrupted by a high priority exception.

enumerator kLPADC_Trigger12InterruptedFlag
Trigger 12 is interrupted by a high priority exception.

enumerator kLPADC_Trigger13InterruptedFlag
Trigger 13 is interrupted by a high priority exception.

enumerator kLPADC_Trigger14InterruptedFlag
Trigger 14 is interrupted by a high priority exception.

enumerator kLPADC_Trigger15InterruptedFlag
Trigger 15 is interrupted by a high priority exception.

enumerator kLPADC_Trigger0CompletedFlag
Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC_Trigger1CompletedFlag
Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enumerator kLPADC_Trigger2CompletedFlag
Trigger 2 is completed and trigger 2 has enabled completion interrupts.

enumerator kLPADC_Trigger3CompletedFlag
Trigger 3 is completed and trigger 3 has enabled completion interrupts.

enumerator kLPADC_Trigger4CompletedFlag
Trigger 4 is completed and trigger 4 has enabled completion interrupts.

enumerator kLPADC_Trigger5CompletedFlag
Trigger 5 is completed and trigger 5 has enabled completion interrupts.

enumerator kLPADC_Trigger6CompletedFlag
Trigger 6 is completed and trigger 6 has enabled completion interrupts.

enumerator kLPADC_Trigger7CompletedFlag
Trigger 7 is completed and trigger 7 has enabled completion interrupts.

enumerator kLPADC_Trigger8CompletedFlag
Trigger 8 is completed and trigger 8 has enabled completion interrupts.

enumerator kLPADC_Trigger9CompletedFlag
Trigger 9 is completed and trigger 9 has enabled completion interrupts.

enumerator kLPADC_Trigger10CompletedFlag
Trigger 10 is completed and trigger 10 has enabled completion interrupts.

enumerator kLPADC_Trigger11CompletedFlag
Trigger 11 is completed and trigger 11 has enabled completion interrupts.

enumerator kLPADC_Trigger12CompletedFlag
Trigger 12 is completed and trigger 12 has enabled completion interrupts.

enumerator kLPADC_Trigger13CompletedFlag
Trigger 13 is completed and trigger 13 has enabled completion interrupts.

enumerator kLPADC_Trigger14CompletedFlag
Trigger 14 is completed and trigger 14 has enabled completion interrupts.

enumerator kLPADC_Trigger15CompletedFlag
Trigger 15 is completed and trigger 15 has enabled completion interrupts.

enum _lpadc_sample_scale_mode
Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Values:

enumerator kLPADC_SamplePartScale
Use divided input voltage signal. (For scale select,please refer to the reference manual).

enumerator kLPADC_SampleFullScale
Full scale (Factor of 1).

enum _lpadc_sample_channel_mode
Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Values:

enumerator kLPADC_SampleChannelSingleEndSideA
Single-end mode, only A-side channel is converted.

enumerator kLPADC_SampleChannelSingleEndSideB
Single-end mode, only B-side channel is converted.

enumerator kLPADC_SampleChannelDiffBothSideAB
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDiffBothSideBA
Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC_SampleChannelDiffBothSide
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDualSingleEndBothSide

Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum _lpadc_hardware_average_mode

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Values:

enumerator kLPADC_HardwareAverageCount1

Single conversion.

enumerator kLPADC_HardwareAverageCount2

2 conversions averaged.

enumerator kLPADC_HardwareAverageCount4

4 conversions averaged.

enumerator kLPADC_HardwareAverageCount8

8 conversions averaged.

enumerator kLPADC_HardwareAverageCount16

16 conversions averaged.

enumerator kLPADC_HardwareAverageCount32

32 conversions averaged.

enumerator kLPADC_HardwareAverageCount64

64 conversions averaged.

enumerator kLPADC_HardwareAverageCount128

128 conversions averaged.

enum _lpadc_sample_time_mode

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Values:

enumerator kLPADC_SampleTimeADCK3

3 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK5

5 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK7

7 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK11

11 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK19
19 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK35
35 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK67
69 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK131
131 ADCK cycles total sample time.

enum _lpadc_hardware_compare_mode
Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Values:

enumerator kLPADC_HardwareCompareDisabled
Compare disabled.

enumerator kLPADC_HardwareCompareStoreOnTrue
Compare enabled. Store on true.

enumerator kLPADC_HardwareCompareRepeatUntilTrue
Compare enabled. Repeat channel acquisition until true.

enum _lpadc_conversion_resolution_mode
Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc_sample_channel_mode_t

Values:

enumerator kLPADC_ConversionResolutionStandard
Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator kLPADC_ConversionResolutionHigh
High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum _lpadc_conversion_average_mode
Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

Values:

enumerator kLPADC_ConversionAverage1
Single conversion.

enumerator kLPADC_ConversionAverage2
2 conversions averaged.

enumerator kLPADC_ConversionAverage4
4 conversions averaged.

enumerator kLPADC_ConversionAverage8
8 conversions averaged.

enumerator kLPADC_ConversionAverage16
16 conversions averaged.

enumerator kLPADC_ConversionAverage32
32 conversions averaged.

enumerator kLPADC_ConversionAverage64
64 conversions averaged.

enumerator kLPADC_ConversionAverage128
128 conversions averaged.

enumerator kLPADC_ConversionAverageMax

enum _lpadc_reference_voltage_mode
Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

Values:

enumerator kLPADC_ReferenceVoltageAlt1
Option 1 setting.

enumerator kLPADC_ReferenceVoltageAlt2
Option 2 setting.

enumerator kLPADC_ReferenceVoltageAlt3
Option 3 setting.

enum _lpadc_power_level_mode
Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Values:

enumerator kLPADC_PowerLevelAlt1
Lowest power setting.

enumerator kLPADC_PowerLevelAlt2
Next lowest power setting.

enumerator kLPADC_PowerLevelAlt3
...

enumerator kLPADC_PowerLevelAlt4
Highest power setting.

enum _lpadc_offset_calibration_mode
Define enumeration of offset calibration mode.

Values:

enumerator kLPADC_OffsetCalibration12bitMode
12 bit offset calibration mode.

enumerator kLPADC_OffsetCalibration16bitMode

16 bit offset calibration mode.

enum _lpadc_trigger_priority_policy

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: kLPADC_TriggerPriorityPreemptSubsequently is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

Values:

enumerator kLPADC_ConvPreemptImmediatelyNotAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator kLPADC_ConvPreemptSoftlyNotAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoRestarted

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptSoftlyAutoRestarted

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator kLPADC_ConvPreemptSoftlyAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator kLPADC_TriggerPriorityPreemptImmediately

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityPreemptSoftly

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityExceptionDisabled

High priority trigger exception disabled.

typedef enum *_lpadc_sample_scale_mode* lpadc_sample_scale_mode_t

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

typedef enum *_lpadc_sample_channel_mode* lpadc_sample_channel_mode_t

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

typedef enum *_lpadc_hardware_average_mode* lpadc_hardware_average_mode_t

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

typedef enum *_lpadc_sample_time_mode* lpadc_sample_time_mode_t

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

typedef enum *_lpadc_hardware_compare_mode* lpadc_hardware_compare_mode_t

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

typedef enum *_lpadc_conversion_resolution_mode* lpadc_conversion_resolution_mode_t

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to lpadc_sample_channel_mode_t

typedef enum *_lpadc_conversion_average_mode* lpadc_conversion_average_mode_t

Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

---

typedef enum *_lpadc_reference_voltage_mode* lpadc_reference_voltage_source_t

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

typedef enum *_lpadc_power_level_mode* lpadc_power_level_mode_t

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

typedef enum *_lpadc_offset_calibration_mode* lpadc_offset_calibration_mode_t

Define enumeration of offset calibration mode.

typedef enum *_lpadc_trigger_priority_policy* lpadc_trigger_priority_policy_t

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

---

typedef struct *_lpadc_calibration_value* lpadc_calibration_value_t

A structure of calibration value.

LPADC_CONVERSION_COMPLETE_TIMEOUT

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

LPADC_CALIBRATION_READY_TIMEOUT

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

LPADC_GAIN_CAL_READY_TIMEOUT

Max loops to wait for LPADC gain calibration GAIN_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN_CAL ready. This parameter defines how many loops to check the gain calibration GAIN_CAL ready. If defined as 0, driver will wait forever until ready.

LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from LPADC_GetStatusFlags().

void LPADC_Init(ADC_Type *base, const *lpadc_config_t* *config)

Initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See "lpadc_config_t".

void LPADC_GetDefaultConfig(*lpadc_config_t* *config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode        = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay            = 0x80;
config->referenceVoltageSource  = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode          = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause         = false;
config->convPauseDelay          = 0U;
config->FIFOWatermark           = 0U;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_Deinit(ADC_Type *base)

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_Enable(ADC_Type *base, bool enable)

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC_DoResetFIFO(ADC_Type *base)

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_DoResetConfig(ADC_Type *base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns

status flags' mask. See to _lpadc_status_flags.

static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clear status flags.

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to _lpadc_status_flags.

static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Parameters

- base – LPADC peripheral base address.

Returns

The OR'ed value of _lpadc_trigger_status_flags.

static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of _lpadc_trigger_status_flags.

static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)

Enable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to _lpadc_interrupt_enable.

static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to _lpadc_interrupt_enable.

static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns

The count of result kept in conversion FIFO.

bool LPADC_GetConvResult(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns
Status whether FIFO entry is valid.

void LPADC_GetConvResultBlocking(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const *lpadc_conv_trigger_config_t* *config)

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to lpadc_conv_trigger_config_t.

void LPADC_GetDefaultConvTriggerConfig(*lpadc_conv_trigger_config_t* *config)

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId       = 0U;
config->delayPower            = 0U;
config->priority              = 0U;
config->channelAFIFOSelect    = 0U;
config->channelBFIFOSelect    = 0U;
config->enableHardwareTrigger = false;
```

Parameters

- config – Pointer to configuration structure.

static inline void LPADC_DoSoftwareTrigger(ADC_Type *base, uint32_t triggerIdMask)

Do software trigger to conversion command.

Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

static inline void LPADC_EnableHardwareTriggerCommandSelection(ADC_Type *base, uint32_t triggerId, bool enable)

Enable hardware trigger command selection.

This function will use the hardware trigger command from ADC_ETC.The trigger command is then defined by ADC hardware trigger command selection field in ADC_ETC->TRIGx_CHAINy_z_n[CSEL].

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- enable – True to enable or flase to disable.

void LPADC_SetConvCommandConfig(ADC_Type *base, uint32_t commandId, const *lpadc_conv_command_config_t* *config)

Configure conversion command.

Note: The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

Parameters

- base – LPADC peripheral base address.
- commandId – ID for command in command buffer. Typically, the available value range is 1 - 15.
- config – Pointer to configuration structure. See to lpadc_conv_command_config_t.

void LPADC_GetDefaultConvCommandConfig(*lpadc_conv_command_config_t* *config)

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode             = kLPADC_SampleFullScale;
config->channelBScaleMode           = kLPADC_SampleFullScale;
config->sampleChannelMode           = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber               = 0U;
config->channelBNumber              = 0U;
config->chainedNextCommandNumber    = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount                   = 0U;
config->hardwareAverageMode         = kLPADC_HardwareAverageCount1;
config->sampleTimeMode              = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode         = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh    = 0U;
config->hardwareCompareValueLow     = 0U;
config->conversionResolutionMode    = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger           = false;
config->enableChannelB              = false;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_EnableCalibration(ADC_Type *base, bool enable)

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6- bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

Parameters

- base – LPADC peripheral base address.
- value – Setting offset value.

status_t LPADC_DoAutoCalibration(ADC_Type *base)

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including: -LPADC_EnableCalibration(...) -LPADC_LPADC_SetOffsetValue(...) -LPADC_SetConvCommandConfig(...) -LPADC_SetConvTriggerConfig(...)

Parameters

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_EnableOffsetCalibration(ADC_Type *base, bool enable)

Enable the offset calibration function.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetCalibrationMode(ADC_Type *base,
lpadc_offset_calibration_mode_t mode)

Set offset calibration mode.

Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to lpadc_offset_calibration_mode_t .

status_t LPADC_DoOffsetCalibration(ADC_Type *base)

Do offset calibration.

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_PrepareAutoCalibration(ADC_Type *base)

Prepare auto calibration, LPADC_FinishAutoCalibration has to be called before using the LPADC. LPADC_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

- base – LPADC peripheral base address.

status_t LPADC_FinishAutoCalibration(ADC_Type *base)

Finish auto calibration start with LPADC_PrepareAutoCalibration.

Note: This feature is used for LPADC with CTRL[CALOFSMODE].

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_GetCalibrationValue(ADC_Type *base, *lpadc_calibration_value_t* *ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

Note: Please note the ADC will be disabled temporary.

Note: This function should be used after finish calibration.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure, this memory block should be always powered on even in low power modes.

status_t LPADC_SetCalibrationValue(ADC_Type *base, const *lpadc_calibration_value_t* *ptrCalibrationValue)

Set calibration value into ADC calibration registers.

Note: Please note the ADC will be disabled temporary.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to lpadc_calibration_value_t structure which contains ADC's calibration value.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.9.4.

struct lpadc_config_t

#include <fsl_lpadc.h> LPADC global configuration.

This structure would used to keep the settings for initialization.

Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay * 4) ADCK cycles must result in a longer delay than the analog startup time.

lpadc_reference_voltage_source_t referenceVoltageSource

Selects the voltage reference high used for conversions.

lpadc_power_level_mode_t powerLevelMode

Power Configuration Selection.

lpadc_trigger_priority_policy_t triggerPriorityPolicy

Control how higher priority triggers are handled, see to lpadc_trigger_priority_policy_t.

bool enableConvPause

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in “Compare Until True” configuration.

uint32_t convPauseDelay

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (convPauseDelay*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

uint32_t FIFOWatermark

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

struct lpadc_conv_command_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion command.

Public Members

lpadc_sample_scale_mode_t sampleScaleMode

Sample scale mode.

lpadc_sample_scale_mode_t channelBScaleMode

Alternate channe B Scale mode.

lpadc_sample_channel_mode_t sampleChannelMode

Channel sample mode.

uint32_t channelNumber

Channel number, select the channel or channel pair.

uint32_t channelBNumber

Alternate Channel B number, select the channel.

uint32_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.

uint32_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

lpadc_hardware_average_mode_t hardwareAverageMode

Hardware average selection.

lpadc_sample_time_mode_t sampleTimeMode

Sample time selection.

lpadc_hardware_compare_mode_t hardwareCompareMode

Hardware compare selection.

uint32_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

lpadc_conversion_resolution_mode_t conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

struct lpadc_conv_trigger_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion trigger.

Public Members

uint32_t targetCommandId

Select the command from command buffer to execute upon detect of the associated trigger event.

uint32_t delayPower

Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is 2^delayPower ADCK cycles. The available value range is 4-bit.

uint32_t priority

Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

bool enableHardwareTrigger

Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. THe software trigger is always available.

struct lpadc_conv_result_t

#include <fsl_lpadc.h> Define the structure to keep the conversion result.

Public Members

uint32_t commandIdSource

Indicate the command buffer being executed that generated this result.

uint32_t loopCountIndex

Indicate the loop count value during command execution that generated this result.

uint32_t triggerIdSource

Indicate the trigger source that initiated a conversion and generated this result.

uint16_t convValue

Data result.

struct _lpadc_calibration_value

#include <fsl_lpadc.h> A structure of calibration value.

2.44 Lpc_freqme

void FREQME_Init(FREQME_Type *base, const *freq_measure_config_t* *config)

Initialize freqme module, set operate mode, operate mode attribute and initialize measurement cycle.

Parameters

- base – FREQME peripheral base address.
- config – The pointer to module basic configuration, please refer to freq_measure_config_t.

void FREQME_GetDefaultConfig(*freq_measure_config_t* *config)

Get default configuration.

```
config->operateMode = kFREQME_FreqMeasurementMode;
config->operateModeAttribute.refClkScaleFactor = 0U;
config->enableContinuousMode                   = false;
config->startMeasurement                       = false;
```

Parameters

- config – The pointer to module basic configuration, please refer to freq_measure_config_t.

static inline void FREQME_StartMeasurementCycle(FREQME_Type *base)

Start frequency or pulse width measurement process.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_TerminateMeasurementCycle(FREQME_Type *base)

Force the termination of any measurement cycle currently in progress and resets RESULT or just reset RESULT if the module in idle state.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_EnableContinuousMode(FREQME_Type *base, bool enable)

Enable/disable Continuous mode.

Parameters

- base – FREQME peripheral base address.
- enable – Used to enable/disable continuous mode,
  - **true** Enable Continuous mode.
  - **false** Disable Continuous mode.

static inline bool FREQME_CheckContinuousMode(FREQME_Type *base)

Check whether continuous mode is enabled.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Continuous mode is enabled, the measurement is performed continuously.
- False – Continuous mode is disabled.

static inline void FREQME_SetOperateMode(FREQME_Type *base, *freqme_operate_mode_t* operateMode)

Set operate mode of freqme module.

Parameters

- base – FREQME peripheral base address.
- operateMode – The operate mode to be set, please refer to freqme_operate_mode_t.

static inline bool FREQME_CheckOperateMode(FREQME_Type *base)

Check module's operate mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Pulse width measurement mode.
- False – Frequency measurement mode.

static inline void FREQME_SetMinExpectedValue(FREQME_Type *base, uint32_t minValue)

Set the minimum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- minValue – The minimum value to set, please note that this value is 31 bits width.

static inline void FREQME_SetMaxExpectedValue(FREQME_Type *base, uint32_t maxValue)

Set the maximum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- maxValue – The maximum value to set, please note that this value is 31 bits width.

uint32_t FREQME_CalculateTargetClkFreq(FREQME_Type *base, uint32_t refClkFrequency)

Calculate the frequency of selected target clock

Note: The formula: Ftarget = (RESULT - 2) * Freference / 2 ^ REF_SCALE.

Note: This function only useful when the operate mode is selected as frequency measurement mode.

Parameters

- base – FREQME peripheral base address.
- refClkFrequency – The frequency of reference clock.

Returns

The frequency of target clock the unit is Hz, if the output result is 0, please check the module's operate mode.

static inline uint8_t FREQME_GetReferenceClkScaleValue(FREQME_Type *base)

Get reference clock scaling factor.

Parameters

- base – FREQME peripheral base address.

Returns

Reference clock scaling factor, the reference count cycle is 2 ^ ref_scale.

static inline void FREQME_SetPulsePolarity(FREQME_Type *base, *freqme_pulse_polarity_t* pulsePolarity)

Set pulse polarity when operate mode is selected as Pulse Width Measurement mode.

Parameters

- base – FREQME peripheral base address.
- pulsePolarity – The pulse polarity to be set, please refer to freqme_pulse_polarity_t.

static inline bool FREQME_CheckPulsePolarity(FREQME_Type *base)

Check pulse polarity when the operate mode is selected as pulse width measurement mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Low period.
- False – High period.

static inline uint32_t FREQME_GetMeasurementResult(FREQME_Type *base)

Get measurement result, if operate mode is selected as pulse width measurement mode this function can be used to calculate pulse width.

Note: Pulse width = counter result / Frequency of target clock.

Parameters

- base – FREQME peripheral base address.

Returns

Measurement result.

static inline uint32_t FREQME_GetInterruptStatusFlags(FREQME_Type *base)

Get interrupt status flags, such as overflow interrupt status flag, underflow interrupt status flag, and so on.

Parameters

- base – FREQME peripheral base address.

Returns

Current interrupt status flags, should be the OR'ed value of _freqme_interrupt_status_flags.

static inline void FREQME_ClearInterruptStatusFlags(FREQME_Type *base, uint32_t statusFlags)

Clear interrupt status flags.

Parameters

- base – FREQME peripheral base address.
- statusFlags – The combination of interrupt status flags to clear, should be the OR'ed value of _freqme_interrupt_status_flags.

static inline void FREQME_EnableInterrupts(FREQME_Type *base, uint32_t masks)

Enable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to enable, should be the OR'ed value of _freqme_interrupt_enable.

static inline void FREQME_DisableInterrupts(FREQME_Type *base, uint32_t masks)

Disable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to disable, should be the OR'ed value of _freqme_interrupt_enable.

FSL_FREQME_DRIVER_VERSION

FREQME driver version 2.1.2.

enum _freqme_interrupt_status_flags

The enumeration of interrupt status flags. .

Values:

enumerator kFREQME_UnderflowInterruptStatusFlag

Indicate the measurement is just done and the result is less than minimun value.

enumerator kFREQME_OverflowInterruptStatusFlag

Indicate the measurement is just done and the result is greater than maximum value.

enumerator kFREQME_ReadyInterruptStatusFlag

Indicate the measurement is just done and the result is ready to read.

enumerator kFREQME_AllInterruptStatusFlags

All interrupt status flags.

enum _freqme_interrupt_enable

The enumeration of interrupts, including underflow interrupt, overflow interrupt, and result ready interrupt. .

Values:

enumerator kFREQME_UnderflowInterruptEnable

Enable interrupt when the result is less than minimum value.

enumerator kFREQME_OverflowInterruptEnable

Enable interrupt when the result is greater than maximum value.

enumerator kFREQME_ReadyInterruptEnable

Enable interrupt when a measurement completes and the result is ready.

enum _freqme_operate_mode

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

Values:

enumerator kFREQME_FreqMeasurementMode

The module works in the frequency measurement mode.

enumerator kFREOME_PulseWidthMeasurementMode

The module works in the pulse width measurement mode.

enum _freqme_pulse_polarity

The enumeration of pulse polarity.

Values:

enumerator kFREQME_PulseHighPeriod

Select high period of the reference clock.

enumerator kFREQME_PulseLowPeriod

Select low period of the reference clock.

typedef enum *_freqme_operate_mode* freqme_operate_mode_t

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

typedef enum *_freqme_pulse_polarity* freqme_pulse_polarity_t

The enumeration of pulse polarity.

typedef union *_freqme_mode_attribute* freqme_mode_attribute_t

The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

---

typedef struct *_freq_measure_config* freq_measure_config_t

The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

union _freqme_mode_attribute

#include <fsl_freqme.h> The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

Public Members

uint8_t refClkScaleFactor

Only useful in frequency measurement operate mode, used to set the reference clock counter scaling factor.

freqme_pulse_polarity_t pulsePolarity

Only Useful in pulse width measurement operate mode, used to set period polarity.

struct _freq_measure_config

#include <fsl_freqme.h> The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

Public Members

freqme_operate_mode_t operateMode

Select operate mode, please refer to freqme_operate_mode_t.

freqme_mode_attribute_t operateModeAttribute

Used to set the attribute of the selected operate mode, if the operate mode is selected as kFREQME_FreqMeasurementMode set freqme_mode_attribute_t::refClkScaleFactor, if operate mode is selected as kFREOME_PulseWidthMeasurementMode, please set freqme_mode_attribute_t::pulsePolarity.

bool enableContinuousMode

Enable/disable continuous mode, if continuous mode is enable, the measurement is performed continuously and the result for the last completed measurement is available in the result register.

2.45 LPFLEXCOMM: LPFLEXCOMM Driver

2.46 LPFLEXCOMM Driver

FSL_LP_FLEXCOMM_DRIVER_VERSION
FlexCOMM driver version.

enum LP_FLEXCOMM_PERIPH_T
LP_FLEXCOMM peripheral modes.

Values:

enumerator LP_FLEXCOMM_PERIPH_NONE
No peripheral

enumerator LP_FLEXCOMM_PERIPH_LPUART
LPUART peripheral

enumerator LP_FLEXCOMM_PERIPH_LPSPI
LPSPI Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2C
LPI2C Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2CAndLPUART
LPI2C and LPUART Peripheral

enum _lpflexcomm_interrupt_flag
LP_FLEXCOMM interrupt source flags.

Values:

enumerator kLPFLEXCOMM_I2cSlaveInterruptFlag

enumerator kLPFLEXCOMM_I2cMasterInterruptFlag

enumerator kLPFLEXCOMM_SpiInterruptFlag

enumerator kLPFLEXCOMM_UartRxInterruptFlag

enumerator kLPFLEXCOMM_UartTxInterruptFlag

enumerator kLPFLEXCOMM_AllInterruptFlag

typedef void (*lpflexcomm_irq_handler_t)(uint32_t instance, void *handle)
Typedef for interrupt handler.

IRQn_Type const kFlexcommIrqs[]
Array with IRQ number for each LP_FLEXCOMM module.

uint32_t LP_FLEXCOMM_GetInstance(void *base)
Returns instance number for LP_FLEXCOMM module with given base address.

uint32_t LP_FLEXCOMM_GetBaseAddress(uint32_t instance)
Returns for LP_FLEXCOMM base address.

uint32_t LP_FLEXCOMM_GetInterruptStatus(uint32_t instance)
brief Returns for LP_FLEXCOMM interrupt source,see _lpflexcomm_interrupt_flag.

status_t LP_FLEXCOMM_Init(uint32_t instance, *LP_FLEXCOMM_PERIPH_T* periph)
Initializes LP_FLEXCOMM and selects peripheral mode according to the second parameter.

void LP_FLEXCOMM_Deinit(uint32_t instance)

Deinitializes LP_FLEXCOMM.

void LP_FLEXCOMM_SetIRQHandler(uint32_t instance, *lpflexcomm_irq_handler_t* handler, void *lpflexcommHandle, *LP_FLEXCOMM_PERIPH_T* periph)

Sets IRQ handler for given LP_FLEXCOMM module. It is used by drivers register IRQ handler according to LP_FLEXCOMM mode.

2.47 LPI2C: Low Power Inter-Integrated Circuit Driver

FSL_LPI2C_DRIVER_VERSION

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy

The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle

The slave driver is idle.

enumerator kStatus_LPI2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail

DMA request failed.

enumerator kStatus_LPI2C_Timeout

Timeout polling status flags.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.48 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster            = true;
masterConfig->debugEnable             = false;
masterConfig->ignoreAck               = false;
masterConfig->pinConfig               = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz             = 100000U;
masterConfig->busIdleTimeout_ns       = 0;
masterConfig->pinLowTimeout_ns        = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable      = false;
masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with LPI2C_MasterInit().

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to lpi2c_master_config_t.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use LPI2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags().

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The kLPI2C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The kLPI2C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like LPI2C_MasterStart(), it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either kLPI2C_Read or kLPI2C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_LPI2C_Nak.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.

- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.

- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferHandleIRQ(uint32_t instance, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- instance – The LPI2C instance.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- kLPI2C_MasterEndOfPacketFlag
- kLPI2C_MasterStopDetectFlag
- kLPI2C_MasterNackDetectFlag
- kLPI2C_MasterArbitrationLostFlag
- kLPI2C_MasterFifoErrFlag
- kLPI2C_MasterPinLowTimeoutFlag
- kLPI2C_MasterDataMatchFlag

All flags except kLPI2C_MasterBusyFlag and kLPI2C_MasterBusBusyFlag can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag
Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag
Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag
End Packet flag

enumerator kLPI2C_MasterStopDetectFlag
Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag
NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag
Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag
FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag
Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag
Data match flag

enumerator kLPI2C_MasterBusyFlag
Master busy flag

enumerator kLPI2C_MasterBusBusyFlag
Bus busy flag

enumerator kLPI2C_MasterClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
Errors to check for.

enum _lpi2c_direction
Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write
Master transmit.

enumerator kLPI2C_Read
Master receive.

enum _lpi2c_master_pin_config
LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain
LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
LPI2C master host request selection.

Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.

Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.

Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the _lpi2c_master_transfer::flags field.

Values:

enumerator kLPI2C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kLPI2C_TransferNoStopFlag
Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t
Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t
LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t
LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t
LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t

LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t

LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterTransferCreateHandle().

Param base
The LPI2C peripheral base address.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(uint32_t instance, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct _lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the LPI2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig
The pin configuration option.

uint32_t baudRate_Hz
Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns
Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns
Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns
Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns
Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest
Host request options.

struct _lpi2c_match_config
#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
Data match configuration setting.

bool rxDataMatchOnly
When set to true, received data is ignored until a successful match.

uint32_t match0
Match value 0.

uint32_t match1
Match value 1.

struct _lpi2c_master_transfer
#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the LPI2C_MasterTransferNonBlocking() API.

Public Members

uint32_t flags
Bit mask of options for the transfer. See enumeration _lpi2c_master_transfer_flags for available options. Set to 0 or kLPI2C_TransferDefaultFlag for normal transfers.

uint16_t slaveAddress
The 7-bit slave address.

lpi2c_direction_t direction
Either kLPI2C_Read or kLPI2C_Write.

uint32_t subaddress
Sub address. Transferred MSB first.

size_t subaddressSize
: Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
: Pointer to data to transfer.

size_t dataSize
: Number of bytes to transfer.

struct __lpi2c_master_handle
: *#include <fsl_lpi2c.h>* Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
: Transfer state machine current state.

uint16_t remainingBytes
: Remaining byte count in current state.

uint8_t *buf
: Buffer pointer for current state.

uint16_t commandBuffer[6]
: LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer
: Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback
: Callback function pointer.

void *userData
: Application data passed to callback.

struct hostRequest

Public Members

bool enable
: Enable host request.

lpi2c_host_request_source_t source
: Host request source.

lpi2c_host_request_polarity_t polarity
: Host request pin polarity.

2.49 LPI2C Master DMA Driver

void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle, *lpi2c_master_edma_transfer_callback_t* callback, void *userData)

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_MasterTransferAbortEDMA() API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

typedef struct *_lpi2c_master_edma_handle* lpi2c_master_edma_handle_t

typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *status_t* completionStatus, void *userData)

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to LPI2C_MasterCreateEDMAHandle().

Param base
The LPI2C peripheral base address.

Param handle
Handle associated with the completed transfer.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

struct _lpi2c_master_edma_handle

#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base

LPI2C base pointer.

bool isBusy

Transfer state machine current state.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[10]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

edma_handle_t *rx

Handle for receive DMA channel.

edma_handle_t *tx

Handle for transmit DMA channel.

edma_tcd_t tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.50 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave              = true;
slaveConfig->address0                 = 0U;
slaveConfig->address1                 = 0U;
slaveConfig->addressMatchMode         = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable         = true;
slaveConfig->filterEnable             = true;
slaveConfig->enableGeneralCall        = false;
slaveConfig->sclStall.enableAck       = false;
slaveConfig->sclStall.enableTx        = true;
slaveConfig->sclStall.enableRx        = true;
slaveConfig->sclStall.enableAddress   = true;
slaveConfig->ignoreAck                = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns  = 0;
slaveConfig->sclGlitchFilterWidth_ns  = 0;
slaveConfig->dataValidDelay_ns        = 0;
slaveConfig->clockHoldTime_ns         = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with LPI2C_SlaveInit(). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t.

void LPI2C_SlaveInit(LPI2C_Type *base, const *lpi2c_slave_config_t* *slaveConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use LPI2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_slave_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag

- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

Attempts to clear other flags has no effect.

See also:

_lpi2c_slave_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags().

static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C slave interrupt requests.

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.

- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the kLPI2C_SlaveTransmitAckFlag is asserted. This only happens if you enable the sclStall.enableAck field of the lpi2c_slave_config_t configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.
- ackOrNack – Pass true for an ACK or false for a NAK.

static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL,false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the kLPI2C_SlaveAddressValidFlag is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns
Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns
Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the LPI2C_SlaveTransferAbort() API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and LPI2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to LPI2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of lpi2c_slave_transfer_event_t enumerators for the events you wish to receive. The kLPI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kLPI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

- eventMask – Bit mask formed by OR'ing together lpi2c_slave_transfer_event_t enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and kLPI2C_SlaveAllEvents to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_LPI2C_Busy – Slave transfers have already been started on this handle.

status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

Return values

- kStatus_Success –
- kStatus_LPI2C_Idle –

void LPI2C_SlaveTransferHandleIRQ(uint32_t instance, void *lpi2cSlaveHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- instance – The LPI2C instance.
- lpi2cSlaveHandle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- kLPI2C_SlaveRepeatedStartDetectFlag
- kLPI2C_SlaveStopDetectFlag
- kLPI2C_SlaveBitErrFlag
- kLPI2C_SlaveFifoErrFlag

All flags except kLPI2C_SlaveBusyFlag and kLPI2C_SlaveBusBusyFlag can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag
Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag
Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag
Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag
Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag
Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag
Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag
Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag
FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag
Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag
Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag
General call flag

enumerator kLPI2C_SlaveBusyFlag
Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag
Bus busy flag

enumerator kLPI2C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags

IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags

Errors to check for.

enum _lpi2c_slave_address_match

LPI2C slave address match options.

Values:

enumerator kLPI2C_MatchAddress0

Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK. When this event is set, the driver will no longer decide to reply to ack/nack.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to LPI2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the LPI2C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct lpi2c_slave_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the LPI2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Enable slave mode.

uint8_t address0

Slave's 7-bit address.

uint8_t address1

Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode
Address matching options.

bool filterDozeEnable
Enable digital glitch filter in doze mode.

bool filterEnable
Enable digital glitch filter.

bool enableGeneralCall
Enable general call address matching.

bool ignoreAck
Continue transfers after a NACK is detected.

bool enableReceivedAddressRead
Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns
Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns
Width in nanoseconds of the clock hold time.

struct _lpi2c_slave_transfer
#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event
Reason the callback is being invoked.

uint8_t receivedAddress
Matching address send by master.

uint8_t *data
Transfer buffer

size_t dataSize
Transfer size

status_t completionStatus
Success or error code describing how the transfer completed. Only applies for kLPI2C_SlaveCompletionEvent.

size_t transferredCount
Number of bytes actually transferred since start or last repeated start.

struct _lpi2c_slave_handle
#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer
: LPI2C slave transfer copy.

bool isBusy
: Whether transfer is busy.

bool wasTransmit
: Whether the last transfer was a transmit.

uint32_t eventMask
: Mask of enabled events.

uint32_t transferredCount
: Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback
: Callback function called at transfer event.

void *userData
: Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck
: Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either 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.

2.51 LPSPI: Low Power Serial Peripheral Interface

2.52 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(uint32_t instance, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- instance – LPSPI instance.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(uint32_t instance, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- instance – LPSPI instance index.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error
LPSPI driver error.

enumerator kStatus_LPSPI_Idle
LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange
LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout
LPSPI timeout polling status flags.

enum _lpspi_flags
LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag
Transmit data flag

enumerator kLPSPI_RxDataReadyFlag
Receive data flag

enumerator kLPSPI_WordCompleteFlag
Word Complete flag

enumerator kLPSPI_FrameCompleteFlag
Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag
Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag
Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag
Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag
Data Match flag

enumerator kLPSPI_ModuleBusyFlag
Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable
LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable
Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0
PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity

LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow

Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow

Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow

Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow

Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow

Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity

LPSPI clock polarity configuration.

Values:

enumerator kLPSPI_ClockPolarityActiveHigh

CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow

CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase

LPSPI clock phase configuration.

Values:

enumerator kLPSPI_ClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction

LPSPI data shifter direction options.

Values:

enumerator kLPSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kLPSPI_LsbFirst

Data transfers start with least significant bit.

enum _lpspi_host_request_select

LPSPI Host Request select configuration.

Values:

enumerator kLPSPI_HostReqExtPin

Host Request is an ext pin.

enumerator kLPSPI_HostReqInternalTrigger
Host Request is an internal trigger.

enum _lpspi_match_config
LPSPI Match configuration options.

Values:

enumerator 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_pcs_function_config
LPSPI cs function configuration.

Values:

enumerator kLPSPI_PcsAsCs
PCS pin select as cs function

enumerator kLPSPI_PcsAsData
PCS pin select as date function

enum _lpspi_transfer_width
LPSPI transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPI delay type selection.

Values:

enumerator kLPSPI_PcsToSck
PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
Delay between transfers.

enum _lpspi_transfer_config_flag_for_master
Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterWidth1
LPSPI master width shift macro, internal used LPSPI master transfer 1bit

enumerator kLPSPI_MasterWidth2
LPSPI master width shift macro, internal used LPSPI master transfer 2bit

enumerator kLPSPI_MasterWidth4
LPSPI master width shift macro, internal used LPSPI master transfer 4bit

enumerator kLPSPI_MasterPcsContinuous
Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap
Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_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_pcs_function_config* lpspi_pcs_function_config_t
LPSPI cs function configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t
LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t
LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t
LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t
LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t
Forward declaration of the _lpspi_master_handle typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t
Forward declaration of the _lpspi_slave_handle typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)
Master completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)
Slave completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_lpspi_transfer* lpspi_transfer_t
LPSPI master/slave transfer structure.

volatile uint8_t g_lpspiDummyData[]
Global variable for dummy data value setting.

LPSPI_DUMMY_DATA
LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

SPI_RETRY_TIMES
Retry times for waiting flag.

LPSPI_MASTER_PCS_SHIFT
LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_PCS_MASK
LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_WIDTH_SHIFT
LPSPI master width shift macro, internal used

LPSPI_MASTER_WIDTH_MASK
LPSPI master width shift mask, internal used

LPSPI_SLAVE_PCS_SHIFT
LPSPI slave PCS shift macro , internal used.

LPSPI_SLAVE_PCS_MASK
LPSPI slave PCS shift macro , internal used.

struct _lpspi_master_config
#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate
Baud Rate for LPSPI.

uint32_t bitsPerFrame
Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
Clock polarity.

lpspi_clock_phase_t cpha
Clock phase.

lpspi_shift_direction_t direction
MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec
: PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec
: Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec
: After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs
: Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
: Desired PCS active high or low

lpspi_pin_config_t pinCfg
: Configures which pins are used for input and output data during single bit transfers.

lpspi_pcs_function_config_t pcsFunc
: Configures cs pins function.

lpspi_data_out_config_t dataOutConfig
: Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay
: Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi__slave__config
: *#include <fsl_lpspi.h>* LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame
: Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
: Clock polarity.

lpspi_clock_phase_t cpha
: Clock phase.

lpspi_shift_direction_t direction
: MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs
: Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
: Desired PCS active high or low

lpspi_pin_config_t pinCfg
: Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
: Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct __lpspi__transfer
: *#include <fsl_lpspi.h>* LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData
Send buffer.

uint8_t *rxData
Receive buffer.

volatile size_t dataSize
Transfer bytes.

uint32_t configFlags
Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

struct __lpspi_master_handle
#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
Is PCS continuous in transfer.

volatile bool writeTcrInIsr
A flag that whether should write TCR in ISR.

volatile bool isByteSwap
A flag that whether should byte swap.

volatile bool isTxMask
A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame
Number of bytes in each frame

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

uint32_t txBuffIfNull
Used if the txData is NULL.

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

struct __lpspi_slave_handle
#include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
A flag that whether should byte swap.

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
: Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

2.53 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION
: LPSPI EDMA driver version.

typedef struct *_lpspi_master_edma_handle* lpspi_master_edma_handle_t
: Forward declaration of the _lpspi_master_edma_handle typedefs.

typedef struct *_lpspi_slave_edma_handle* lpspi_slave_edma_handle_t
: Forward declaration of the _lpspi_slave_edma_handle typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)
: Completion callback function pointer type.

  Param base
  : LPSPI peripheral base address.

  Param handle
  : Pointer to the handle for the LPSPI master.

  Param status
  : Success or error code describing whether the transfer completed.

  Param userData
  : Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)
: Completion callback function pointer type.

  Param base
  : LPSPI peripheral base address.

  Param handle
  : Pointer to the handle for the LPSPI slave.

  Param status
  : Success or error code describing whether the transfer completed.

  Param userData
  : Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- configFlags – transfer configuration flags. _lpspi_transfer_config_flag_for_master.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

Returns
Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure, config field is not 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.

uint32_t totalByteCount

Number of transfer bytes

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint32_t txBuffIfNull

Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull

Used if there is not rxData for DMA purpose.

uint32_t transmitCommand

Used to write TCR for DMA purpose.

volatile int8_t oneFifoBlockRxWatermark

Used to change RXWATER in FCR for DMA purpose.

volatile int8_t lastBytesRxWatermark

Used to change RXWATER in FCR for DMA purpose.

volatile uint8_t state

LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_edma_transfer_callback_t callback

Completion callback.

void *userData

Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[6]
SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
#include <fsl_lpspi_edma.h> LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

uint32_t totalByteCount
Number of transfer bytes

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint32_t txBuffIfNull
Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
Used if there is not rxData for DMA purpose.

volatile int8_t oneFifoBlockRxWatermark
Used to change RXWATER in FCR for DMA purpose.

volatile int8_t lastBytesRxWatermark
Used to change RXWATER in FCR for DMA purpose.

volatile uint8_t state
LPSPI transfer state.

lpspi_slave_edma_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg

edma_tcd_t lpspiSoftwareTCD[5]
SoftwareTCD, internal used

2.54 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.55 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

void LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, 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_NoiseErrorInRxDataRegFlag, kLPUART_ParityErrorInRxDataRegFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the _lpuart_interrupt_enable. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of the enumeration _uart_interrupt_enable.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _lpuart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators _lpuart_interrupt_enable. To check a specific interrupt enable status, compare the return value with enumerators in _lpuart_interrupt_enable. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns
LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

static inline uint32_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns
LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns
LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the 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(uint32_t instance, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- instance – LPUART instance.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually
Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.

Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.

Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.

Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select
LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_CtsStateChangeInterruptEnable
Change of state on CTS_B pin. bit 0

enumerator kLPUART_DsrStateChangeInterruptEnable
Change of state on DSR_B pin. bit 1

enumerator kLPUART_RinStateChangeInterruptEnable
Change of state on RIN_B pin. bit 2

enumerator kLPUART_DcdStateChangeInterruptEnable
Change of state on DCD_B pin. bit 3

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxCounter0TimeoutInterruptEnable
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutInterruptEnable
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutInterruptEnable
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutInterruptEnable
Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2InterruptEnable
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1InterruptEnable
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_CtsStateChangeFlag
Change of state on CTS_B pin. bit 2

enumerator kLPUART_DsrStateChangeFlag
Change of state on DSR_B pin. bit 3

enumerator kLPUART_RinStateChangeFlag
Change of state on RIN_B pin. bit 4

enumerator kLPUART_DcdStateChangeFlag
Change of state on DCD_B pin. bit 5

enumerator kLPUART_RxCounter0TimeoutFlag
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutFlag
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutFlag
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutFlag
Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_TxDataRegEmptyFlag

Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

enum _lpuart_timeout_condition

LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

Values:

enumerator kLPUART_TimeoutAfterCharacters

Timeout occurs when the number of characters specified by timeoutValue are received.

enumerator kLPUART_TimeoutAfterIdle

Timeout occurs when rx/tx remains idle for timeoutValue of bit clocks after idle condition is detected.

enumerator kLPUART_TimeoutAfterNext

Timeout occurs when rx/tx remains idle for timeoutValue of bit clocks after next character is received/transmitted.

enumerator kLPUART_TimeoutAfterIdleBeforeExtended

Timeout occurs when tx/rx is idle for larger than timeoutValue of bit clocks and smaller than tx/rx extended timeout value.

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
LPUART transmit CTS configure.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef enum *_lpuart_timeout_condition* lpuart_timeout_condition_t
LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

typedef struct *_lpuart_timeout_counter_config* lpuart_timeout_counter_config_t
LPUART timeout counter configuration structure.

typedef struct *_lpuart_timeout_config* lpuart_timeout_config_t
LPUART timeout configuration structure.

typedef struct *_lpuart_config* lpuart_config_t
LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
LPUART transfer callback function.

typedef void (*lpuart_irq_handler_t)(uint32_t instance, void *handle)

const IRQn_Type s_lpuartIRQ[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_timeout_counter_config
#include <fsl_lpuart.h> LPUART timeout counter configuration structure.

Public Members

bool enableCounter
Eneble the timeout counter.

lpuart_timeout_condition_t timeoutCondition
Timeout condition.

uint16_t timeoutValue
Timeout value.

struct _lpuart_timeout_config
#include <fsl_lpuart.h> LPUART timeout configuration structure.

Public Members

uint16_t rxExtendedTimeoutValue
The number of bits since the last stop bit that is required for an idle condition to be detected. Enable this will disable rxIdleType and rxIdleConfig. Set to 0 to disable.

uint16_t txExtendedTimeoutValue
The transmitter idle time in number of bits (baud rate) whenever an idle character is queued through the transmit FIFO.

lpuart_timeout_counter_config_t rxCounter0
Rx counter 0 configuration.

lpuart_timeout_counter_config_t rxCounter1
Rx counter 1 configuration.

lpuart_timeout_counter_config_t txCounter0
Tx counter 0 configuration.

lpuart_timeout_counter_config_t txCounter1
Tx counter 1 configuration.

struct _lpuart_config
#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
RX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

lpuart_timeout_config_t timeoutConfig
Timeout configuration.

bool enableSingleWire
Use TXD pin as the source for the receiver. When enabled the TXD pin should be configured as open drain.

uint8_t rtsDelay
Delay the negation of RTS by the configured number of bit clocks.

bool enableTx
Enable TX

bool enableRx
Enable RX

struct __lpuart__transfer
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __lpuart__handle
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

volatile size_t txDataSize
Size of the remaining data to send.

size_t txDataSizeAll
Size of the data to send out.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t rxDataSizeAll
Size of the data to receive.

size_t rxRingBufferSize
Size of the ring buffer.

volatile uint16_t rxRingBufferHead
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
: Callback function.

void *userData
: LPUART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state.

bool isSevenDataBits
: Seven data bits flag.

bool is16bitData
: 16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed78___

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 ___unnamed80___

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 ___unnamed82___

Public Members

uint8_t *volatile rxData
: Address of remaining data to receive.

uint16_t *volatile rxData16
: Address of remaining data to receive.

union ___unnamed84___

Public Members

uint8_t *rxRingBuffer

Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16

Start address of the receiver ring buffer.

2.56 LPUART eDMA Driver

void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

status_t LPUART_SendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferEdmaHandleIRQ(uint32_t instance, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- instance – LPUART peripheral index.
- lpuartEdmaHandle – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

typedef struct *_lpuart_edma_handle* lpuart_edma_handle_t

typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

struct _lpuart_edma_handle

#include <fsl_lpuart_edma.h> LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback

Callback function.

void *userData

LPUART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.57 MIPI DSI Driver

void DSI_Init(MIPI_DSI_HOST_Type *base, const *dsi_config_t* *config)

Initializes an MIPI DSI host with the user configuration.

This function initializes the MIPI DSI host with the configuration, it should be called first before other MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to a user-defined configuration structure.

void DSI_Deinit(MIPI_DSI_HOST_Type *base)

Deinitializes an MIPI DSI host.

This function should be called after all bother MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.

void DSI_GetDefaultConfig(*dsi_config_t* *config)

Get the default configuration to initialize the MIPI DSI host.

The default value is:

```
config->numLanes = 4;
config->enableNonContinuousHsClk = false;
config->enableTxUlps = false;
config->autoInsertEoTp = true;
config->numExtraEoTp = 0;
config->htxTo_ByteClk = 0;
config->lrxHostTo_ByteClk = 0;
config->btaTo_ByteClk = 0;
```

Parameters

- config – Pointer to a user-defined configuration structure.

void DSI_SetDpiConfig(MIPI_DSI_HOST_Type *base, const *dsi_dpi_config_t* *config, uint8_t numLanes, uint32_t dpiPixelClkFreq_Hz, uint32_t dsiHsBitClkFreq_Hz)

Configure the DPI interface core.

This function sets the DPI interface configuration, it should be used in video mode.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the DPI interface configuration.
- numLanes – Lane number, should be same with the setting in dsi_dpi_config_t.
- dpiPixelClkFreq_Hz – The DPI pixel clock frequency in Hz.
- dsiHsBitClkFreq_Hz – The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output.

uint32_t DSI_InitDphy(MIPI_DSI_HOST_Type *base, const *dsi_dphy_config_t* *config, uint32_t refClkFreq_Hz)

Initializes the D-PHY.

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The configuration structure could be got by the function DSI_GetDphyDefaultConfig.

For some platforms there is not dedicated D-PHY PLL, indicated by the macro FSL_FEATURE_MIPI_DSI_NO_DPHY_PLL. For these platforms, the refClkFreq_Hz is useless.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the D-PHY configuration.
- refClkFreq_Hz – The REFCLK frequency in Hz.

Returns

The actual D-PHY PLL output frequency. If could not configure the PLL to the target frequency, the return value is 0.

void DSI_DeinitDphy(MIPI_DSI_HOST_Type *base)

Deinitializes the D-PHY.

Power down the D-PHY PLL and shut down D-PHY.

Parameters

- base – MIPI DSI host peripheral base address.

void DSI_GetDphyDefaultConfig(*dsi_dphy_config_t* *config, uint32_t txHsBitClk_Hz, uint32_t txEscClk_Hz)

Get the default D-PHY configuration.

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User could use the configuration directly, or change some parameters according to the special device.

Parameters

- config – Pointer to the D-PHY configuration.
- txHsBitClk_Hz – High speed bit clock in Hz.
- txEscClk_Hz – Esc clock in Hz.

static inline void DSI_SetUlpsStatus(MIPI_DSI_HOST_Type *base, uint8_t ulpsStatus)

Sets the ULPS status for one clock lane and 4 data lanes.

Use status in _dsi_ulps_status as OR'ed mask value for ulpsStatus. Release the lanes from ULPS state before transfer.

Parameters

- base – MIPI DSI host peripheral base address.
- ulpsStatus – The ULPS status to set to the bus.

static inline uint8_t DSI_GetUlpsStatus(MIPI_DSI_HOST_Type *base)

Gets the ULPS status for one clock lane and 4 data lanes.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The current ULPS status.

static inline void DSI_EnableInterrupts(MIPI_DSI_HOST_Type *base, uint32_t intGroup1, uint32_t intGroup2)

Enable the interrupts.

The interrupts to enable are passed in as OR'ed mask value of _dsi_interrupt.

Parameters

- base – MIPI DSI host peripheral base address.

- intGroup1 – Interrupts to enable in group 1.
- intGroup2 – Interrupts to enable in group 2.

static inline void DSI_DisableInterrupts(MIPI_DSI_HOST_Type *base, uint32_t intGroup1, uint32_t intGroup2)

Disable the interrupts.

The interrupts to disable are passed in as OR'ed mask value of _dsi_interrupt.

Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Interrupts to disable in group 1.
- intGroup2 – Interrupts to disable in group 2.

static inline void DSI_GetAndClearInterruptStatus(MIPI_DSI_HOST_Type *base, uint32_t *intGroup1, uint32_t *intGroup2)

Get and clear the interrupt status.

Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Group 1 interrupt status.
- intGroup2 – Group 2 interrupt status.

static inline void DSI_SetDbiPixelFormat(MIPI_DSI_HOST_Type *base, *dsi_dbi_pixel_format_t* format)

brief Configure the DBI pixel format.

param base MIPI DSI host peripheral base address. param format of the pixel.

static inline void DSI_SetDbiPixelFifoSendLevel(MIPI_DSI_HOST_Type *base, uint16_t sendLevel)

Configure the DBI pixel FIFO send level.

This controls the level at which the DBI Host bridge begins sending pixels

Parameters

- base – MIPI DSI host peripheral base address.
- sendLevel – Send level value set to register.

static inline void DSI_SetDbiPixelPayloadSize(MIPI_DSI_HOST_Type *base, uint16_t payloadSize)

Configure the DBI pixel payload size.

Configures maximum number of pixels that should be sent as one DSI packet. Recommended to be evenly divisible by the line size (in pixels).

Parameters

- base – MIPI DSI host peripheral base address.
- payloadSize – Payload size value set to register.

void DSI_SetApbPacketControl(MIPI_DSI_HOST_Type *base, uint16_t wordCount, uint8_t virtualChannel, *dsi_tx_data_type_t* dataType, uint8_t flags)

Configure the APB packet to send.

This function configures the next APB packet transfer. After configuration, the packet transfer could be started with function DSI_SendApbPacket. If the packet is long packet, Use DSI_WriteApbTxPayload to fill the payload before start transfer.

Parameters

- base – MIPI DSI host peripheral base address.
- wordCount – For long packet, this is the byte count of the payload. For short packet, this is (data1 « 8) | data0.
- virtualChannel – Virtual channel.
- dataType – The packet data type, (DI).
- flags – The transfer control flags, see _dsi_transfer_flags.

void DSI_WriteApbTxPayload(MIPI_DSI_HOST_Type *base, const uint8_t *payload, uint16_t payloadSize)

Fill the long APB packet payload.

Write the long packet payload to TX FIFO.

Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.

void DSI_WriteApbTxPayloadExt(MIPI_DSI_HOST_Type *base, const uint8_t *payload, uint16_t payloadSize, bool sendDcsCmd, uint8_t dcsCmd)

Extended function to fill the payload to TX FIFO.

Write the long packet payload to TX FIFO. This function could be used in two ways

a. Include the DCS command in parameter payload. In this case, the DCS command is the first byte of payload. The parameter sendDcsCmd is set to false, the dcsCmd is not used. This function is the same as DSI_WriteApbTxPayload when used in this way.
b. The DCS command in not in parameter payload, but specified by parameter dcsCmd. In this case, the parameter sendDcsCmd is set to true, the dcsCmd is the DCS command to send. The payload is sent after dcsCmd.

Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.
- sendDcsCmd – If set to true, the DCS command is specified by dcsCmd, otherwise the DCS command is included in the payload.
- dcsCmd – The DCS command to send, only used when sendDCSCmd is true.

void DSI_ReadApbRxPayload(MIPI_DSI_HOST_Type *base, uint8_t *payload, uint16_t payloadSize)

Read the long APB packet payload.

Read the long packet payload from RX FIFO. This function reads directly but does not check the RX FIFO status. Upper layer should make sure there are available data.

Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.

static inline void DSI_SendApbPacket(MIPI_DSI_HOST_Type *base)

Trigger the controller to send out APB packet.

Send the packet set by DSI_SetApbPacketControl.

Parameters

- base – MIPI DSI host peripheral base address.

static inline uint32_t DSI_GetApbStatus(MIPI_DSI_HOST_Type *base)

Get the APB status.

The return value is OR'ed value of _dsi_apb_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The APB status.

static inline uint32_t DSI_GetRxErrorStatus(MIPI_DSI_HOST_Type *base)

Get the error status during data transfer.

The return value is OR'ed value of _dsi_rx_error_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The error status.

static inline uint8_t DSI_GetEccRxErrorPosition(uint32_t rxErrorStatus)

Get the one-bit RX ECC error position.

When one-bit ECC RX error detected using DSI_GetRxErrorStatus, this function could be used to get the error bit position.

```
uint8_t eccErrorPos;
uint32_t rxErrorStatus = DSI_GetRxErrorStatus(MIPI_DSI);
if (kDSI_RxErrorEccOneBit & rxErrorStatus)
{
    eccErrorPos = DSI_GetEccRxErrorPosition(rxErrorStatus);
}
```

Parameters

- rxErrorStatus – The error status returned by DSI_GetRxErrorStatus.

Returns

The 1-bit ECC error position.

static inline uint32_t DSI_GetAndClearHostStatus(MIPI_DSI_HOST_Type *base)

Get and clear the DSI host status.

The host status are returned as mask value of _dsi_host_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The DSI host status.

static inline uint32_t DSI_GetRxPacketHeader(MIPI_DSI_HOST_Type *base)

Get the RX packet header.

Parameters

- base – MIPI DSI host peripheral base address.

Returns
The RX packet header.

static inline *dsi_rx_data_type_t* DSI_GetRxPacketType(uint32_t rxPktHeader)

Extract the RX packet type from the packet header.

Extract the RX packet type from the packet header get by DSI_GetRxPacketHeader.

Parameters

- rxPktHeader – The RX packet header get by DSI_GetRxPacketHeader.

Returns
The RX packet type.

static inline uint16_t DSI_GetRxPacketWordCount(uint32_t rxPktHeader)

Extract the RX packet word count from the packet header.

Extract the RX packet word count from the packet header get by DSI_GetRxPacketHeader.

Parameters

- rxPktHeader – The RX packet header get by DSI_GetRxPacketHeader.

Returns
For long packet, return the payload word count (byte). For short packet, return the (data0 « 8) | data1.

static inline uint8_t DSI_GetRxPacketVirtualChannel(uint32_t rxPktHeader)

Extract the RX packet virtual channel from the packet header.

Extract the RX packet virtual channel from the packet header get by DSI_GetRxPacketHeader.

Parameters

- rxPktHeader – The RX packet header get by DSI_GetRxPacketHeader.

Returns
The virtual channel.

status_t DSI_TransferBlocking(MIPI_DSI_HOST_Type *base, *dsi_transfer_t* *xfer)

APB data transfer using blocking method.

Perform APB data transfer using blocking method. This function waits until all data send or received, or timeout happens.

When using this API to read data, the actually read data count could be got from xfer->rxDataSize.

Parameters

- base – MIPI DSI host peripheral base address.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data transfer finished with no error.
- kStatus_Timeout – Transfer failed because of timeout.
- kStatus_DSI_RxDataError – RX data error, user could use DSI_GetRxErrorStatus to check the error details.
- kStatus_DSI_ErrorReportReceived – Error Report packet received, user could use DSI_GetAndClearHostStatus to check the error report status.
- kStatus_DSI_NotSupported – Transfer format not supported.

- kStatus_DSI_Fail – Transfer failed for other reasons.

status_t DSI_TransferCreateHandle(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle, *dsi_callback_t* callback, void *userData)

Create the MIPI DSI handle.

This function initializes the MIPI DSI handle which can be used for other transactional APIs.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – Handle pointer.
- callback – Callback function.
- userData – User data.

status_t DSI_TransferNonBlocking(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle, *dsi_transfer_t* *xfer)

APB data transfer using interrupt method.

Perform APB data transfer using interrupt method, when transfer finished, upper layer could be informed through callback function.

When using this API to read data, the actually read data count could be got from handle->xfer->rxDataSize after read finished.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data transfer started successfully.
- kStatus_DSI_Busy – Failed to start transfer because DSI is busy with pervious transfer.
- kStatus_DSI_NotSupported – Transfer format not supported.

void DSI_TransferAbort(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle)

Abort current APB data transfer.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.

void DSI_TransferHandleIRQ(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle)

Interrupt handler for the DSI.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.

FSL_MIPI_DSI_DRIVER_VERSION

Error codes for the MIPI DSI driver.

Values:

enumerator kStatus_DSI_Busy
DSI is busy.

enumerator kStatus_DSI_RxDataError
Read data error.

enumerator kStatus_DSI_ErrorReportReceived
Error report package received.

enumerator kStatus_DSI_NotSupported
The transfer type not supported.

enum _dsi_dbi_pixel_format
MIPI DBI pixel format.

Values:

enumerator kDSI_DbiNoMapping
No pixel to bit mapping.

enumerator kDSI_DbiRGB888
Send data to interface as RGB888 format.

enumerator kDSI_DbiRGB666
Send data to interface as RGB666 format.

enumerator kDSI_DbiRGB565
Send data to interface as RGB565 format.

enumerator kDSI_DbiRGB444
Send data to interface as RGB444 format.

enumerator kDSI_DbiRGB332
Send data to interface as RGB332 format.

enum _dsi_dpi_color_coding
MIPI DPI interface color coding.

Values:

enumerator kDSI_Dpi16BitConfig1
16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator kDSI_Dpi16BitConfig2
16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator kDSI_Dpi16BitConfig3
16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator kDSI_Dpi18BitConfig1
18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator kDSI_Dpi18BitConfig2
18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator kDSI_Dpi24Bit
24-bit.

enum _dsi_dpi_pixel_packet
MIPI DSI pixel packet type send through DPI interface.

Values:

enumerator kDSI_PixelPacket16Bit
16 bit RGB565.

enumerator kDSI_PixelPacket18Bit
18 bit RGB666 packed.

enumerator kDSI_PixelPacket18BitLoosely
18 bit RGB666 loosely packed into three bytes.

enumerator kDSI_PixelPacket24Bit
24 bit RGB888, each pixel uses three bytes.

_dsi_dpi_polarity_flag DPI signal polarity.

Values:

enumerator kDSI_DpiVsyncActiveLow
VSYNC active low.

enumerator kDSI_DpiHsyncActiveLow
HSYNC active low.

enumerator kDSI_DpiVsyncActiveHigh
VSYNC active high.

enumerator kDSI_DpiHsyncActiveHigh
HSYNC active high.

enum _dsi_dpi_video_mode
DPI video mode.

Values:

enumerator kDSI_DpiNonBurstWithSyncPulse
Non-Burst mode with Sync Pulses.

enumerator kDSI_DpiNonBurstWithSyncEvent
Non-Burst mode with Sync Events.

enumerator kDSI_DpiBurst
Burst mode.

enum _dsi_dpi_bllp_mode
Behavior in BLLP (Blanking or Low-Power Interval).

Values:

enumerator kDSI_DpiBllpLowPower
LP mode used in BLLP periods.

enumerator kDSI_DpiBllpBlanking
Blanking packets used in BLLP periods.

enumerator kDSI_DpiBllpNull
Null packets used in BLLP periods.

_dsi_apb_status Status of APB to packet interface.

Values:

enumerator kDSI_ApbNotIdle
State machine not idle

enumerator kDSI_ApbTxDone
Tx packet done

enumerator kDSI_ApbRxControl
DPHY direction 0 - tx had control, 1 - rx has control

enumerator kDSI_ApbTxOverflow
TX fifo overflow

enumerator kDSI_ApbTxUnderflow
TX fifo underflow

enumerator kDSI_ApbRxOverflow
RX fifo overflow

enumerator kDSI_ApbRxUnderflow
RX fifo underflow

enumerator kDSI_ApbRxHeaderReceived
RX packet header has been received

enumerator kDSI_ApbRxPacketReceived
All RX packet payload data has been received

_dsi_rx_error_status Host receive error status.

Values:

enumerator kDSI_RxErrorEccOneBit
ECC single bit error detected.

enumerator kDSI_RxErrorEccMultiBit
ECC multi bit error detected.

enumerator kDSI_RxErrorCrc
CRC error detected.

enumerator kDSI_RxErrorHtxTo
High Speed forward TX timeout detected.

enumerator kDSI_RxErrorLrxTo
Reverse Low power data receive timeout detected.

enumerator kDSI_RxErrorBtaTo
BTA timeout detected.

enum _dsi_host_status
DSI host controller status (status_out)

Values:

enumerator kDSI_HostSoTError
SoT error from peripheral error report.

enumerator kDSI_HostSoTSyncError
SoT Sync error from peripheral error report.

enumerator kDSI_HostEoTSyncError
EoT Sync error from peripheral error report.

enumerator kDSI_HostEscEntryCmdError
Escape Mode Entry Command Error from peripheral error report.

enumerator kDSI_HostLpTxSyncError
Low-power transmit Sync Error from peripheral error report.

enumerator kDSI_HostPeriphToError
Peripheral timeout error from peripheral error report.

enumerator kDSI_HostFalseControlError
False control error from peripheral error report.

enumerator kDSI_HostContentionDetected
Contention detected from peripheral error report.

enumerator kDSI_HostEccErrorOneBit
Single bit ECC error (corrected) from peripheral error report.

enumerator kDSI_HostEccErrorMultiBit
Multi bit ECC error (not corrected) from peripheral error report.

enumerator kDSI_HostChecksumError
Checksum error from peripheral error report.

enumerator kDSI_HostInvalidDataType
DSI data type not recognized.

enumerator kDSI_HostInvalidVcId
DSI VC ID invalid.

enumerator kDSI_HostInvalidTxLength
Invalid transmission length.

enumerator kDSI_HostProtocalViolation
DSI protocal violation.

enumerator kDSI_HostResetTriggerReceived
Reset trigger received.

enumerator kDSI_HostTearTriggerReceived
Tear effect trigger receive.

enumerator kDSI_HostAckTriggerReceived
Acknowledge trigger message received.

_dsi_ulps_status Status of APB to packet interface.

Values:

enumerator kDSI_ClockLaneUlpsEnable
Clock lane in ULPS mode.

enumerator kDSI_DataLane0UlpsEnable
Data lane 0 in ULPS mode.

enumerator kDSI_DataLane1UlpsEnable
Data lane 1 in ULPS mode.

enumerator kDSI_DataLane2UlpsEnable
Data lane 2 in ULPS mode.

enumerator kDSI_DataLane3UlpsEnable
Data lane 3 in ULPS mode.

_dsi_interrupt DSI interrupt.

Values:

enumerator kDSI_InterruptGroup1ApbNotIdle
State machine not idle

enumerator kDSI_InterruptGroup1ApbTxDone
Tx packet done

enumerator kDSI_InterruptGroup1ApbRxControl
DPHY direction 0 - tx control, 1 - rx control

enumerator kDSI_InterruptGroup1ApbTxOverflow
TX fifo overflow

enumerator kDSI_InterruptGroup1ApbTxUnderflow
TX fifo underflow

enumerator kDSI_InterruptGroup1ApbRxOverflow
RX fifo overflow

enumerator kDSI_InterruptGroup1ApbRxUnderflow
RX fifo underflow

enumerator kDSI_InterruptGroup1ApbRxHeaderReceived
RX packet header has been received

enumerator kDSI_InterruptGroup1ApbRxPacketReceived
All RX packet payload data has been received

enumerator kDSI_InterruptGroup1SoTError
SoT error from peripheral error report.

enumerator kDSI_InterruptGroup1SoTSyncError
SoT Sync error from peripheral error report.

enumerator kDSI_InterruptGroup1EoTSyncError
EoT Sync error from peripheral error report.

enumerator kDSI_InterruptGroup1EscEntryCmdError
Escape Mode Entry Command Error from peripheral error report.

enumerator kDSI_InterruptGroup1LpTxSyncError
Low-power transmit Sync Error from peripheral error report.

enumerator kDSI_InterruptGroup1PeriphToError
Peripheral timeout error from peripheral error report.

enumerator kDSI_InterruptGroup1FalseControlError
False control error from peripheral error report.

enumerator kDSI_InterruptGroup1ContentionDetected
Contention detected from peripheral error report.

enumerator kDSI_InterruptGroup1EccErrorOneBit
Single bit ECC error (corrected) from peripheral error report.

enumerator kDSI_InterruptGroup1EccErrorMultiBit
Multi bit ECC error (not corrected) from peripheral error report.

enumerator kDSI_InterruptGroup1ChecksumError
Checksum error from peripheral error report.

enumerator kDSI_InterruptGroup1InvalidDataType
DSI data type not recognized.

enumerator kDSI_InterruptGroup1InvalidVcId
DSI VC ID invalid.

enumerator kDSI_InterruptGroup1InvalidTxLength
Invalid transmission length.

enumerator kDSI_InterruptGroup1ProtocalViolation
DSI protocal violation.

enumerator kDSI_InterruptGroup1ResetTriggerReceived
Reset trigger received.

enumerator kDSI_InterruptGroup1TearTriggerReceived
Tear effect trigger receive.

enumerator kDSI_InterruptGroup1AckTriggerReceived
Acknowledge trigger message received.

enumerator kDSI_InterruptGroup1HtxTo
High speed TX timeout.

enumerator kDSI_InterruptGroup1LrxTo
Low power RX timeout.

enumerator kDSI_InterruptGroup1BtaTo
Host BTA timeout.

enumerator kDSI_InterruptGroup2EccOneBit
Sinle bit ECC error.

enumerator kDSI_InterruptGroup2EccMultiBit
Multi bit ECC error.

enumerator kDSI_InterruptGroup2CrcError
CRC error.

enum _dsi_tx_data_type
DSI TX data type.

Values:

enumerator kDSI_TxDataVsyncStart
V Sync start.

enumerator kDSI_TxDataVsyncEnd
V Sync end.

enumerator kDSI_TxDataHsyncStart
H Sync start.

enumerator kDSI_TxDataHsyncEnd
H Sync end.

enumerator kDSI_TxDataEoTp
End of transmission packet.

enumerator kDSI_TxDataCmOff
Color mode off.

enumerator kDSI_TxDataCmOn
Color mode on.

enumerator kDSI_TxDataShutDownPeriph
Shut down peripheral.

enumerator kDSI_TxDataTurnOnPeriph
Turn on peripheral.

enumerator kDSI_TxDataGenShortWrNoParam
Generic Short WRITE, no parameters.

enumerator kDSI_TxDataGenShortWrOneParam
Generic Short WRITE, one parameter.

enumerator kDSI_TxDataGenShortWrTwoParam
Generic Short WRITE, two parameter.

enumerator kDSI_TxDataGenShortRdNoParam
Generic Short READ, no parameters.

enumerator kDSI_TxDataGenShortRdOneParam
Generic Short READ, one parameter.

enumerator kDSI_TxDataGenShortRdTwoParam
Generic Short READ, two parameter.

enumerator kDSI_TxDataDcsShortWrNoParam
DCS Short WRITE, no parameters.

enumerator kDSI_TxDataDcsShortWrOneParam
DCS Short WRITE, one parameter.

enumerator kDSI_TxDataDcsShortRdNoParam
DCS Short READ, no parameters.

enumerator kDSI_TxDataSetMaxReturnPktSize
Set the Maximum Return Packet Size.

enumerator kDSI_TxDataNull
Null Packet, no data.

enumerator kDSI_TxDataBlanking
Blanking Packet, no data.

enumerator kDSI_TxDataGenLongWr
Generic long write.

enumerator kDSI_TxDataDcsLongWr
DCS Long Write/write_LUT Command Packet.

enumerator kDSI_TxDataLooselyPackedPixel20BitYCbCr
Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel24BitYCbCr
Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel16BitYCbCr
Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel30BitRGB
Packed Pixel Stream, 30-bit RGB, 10-10-10 Format.

enumerator kDSI_TxDataPackedPixel36BitRGB
Packed Pixel Stream, 36-bit RGB, 12-12-12 Format.

enumerator kDSI_TxDataPackedPixel12BitYCrCb
Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format.

enumerator kDSI_TxDataPackedPixel16BitRGB
Packed Pixel Stream, 16-bit RGB, 5-6-5 Format.

enumerator kDSI_TxDataPackedPixel18BitRGB
Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataLooselyPackedPixel18BitRGB
Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataPackedPixel24BitRGB
Packed Pixel Stream, 24-bit RGB, 8-8-8 Format.

enum _dsi_rx_data_type
DSI RX data type.

Values:

enumerator kDSI_RxDataAckAndErrorReport
Acknowledge and Error Report

enumerator kDSI_RxDataEoTp
End of Transmission packet.

enumerator kDSI_RxDataGenShortRdResponseOneByte
Generic Short READ Response, 1 byte returned.

enumerator kDSI_RxDataGenShortRdResponseTwoByte
Generic Short READ Response, 2 byte returned.

enumerator kDSI_RxDataGenLongRdResponse
Generic Long READ Response.

enumerator kDSI_RxDataDcsLongRdResponse
DCS Long READ Response.

enumerator kDSI_RxDataDcsShortRdResponseOneByte
DCS Short READ Response, 1 byte returned.

enumerator kDSI_RxDataDcsShortRdResponseTwoByte
DCS Short READ Response, 2 byte returned.

_dsi_transfer_flags DSI transfer control flags.

Values:

enumerator kDSI_TransferUseHighSpeed
Use high speed mode or not.

enumerator kDSI_TransferPerformBTA
Perform BTA or not.

typedef enum *_dsi_dbi_pixel_format* dsi_dbi_pixel_format_t
MIPI DBI pixel format.

typedef struct *_dsi_config* dsi_config_t
MIPI DSI controller configuration.

typedef enum *_dsi_dpi_color_coding* dsi_dpi_color_coding_t
MIPI DPI interface color coding.

typedef enum *_dsi_dpi_pixel_packet* dsi_dpi_pixel_packet_t
MIPI DSI pixel packet type send through DPI interface.

typedef enum *_dsi_dpi_video_mode* dsi_dpi_video_mode_t
DPI video mode.

typedef enum *_dsi_dpi_bllp_mode* dsi_dpi_bllp_mode_t
Behavior in BLLP (Blanking or Low-Power Interval).

typedef struct *_dsi_dpi_config* dsi_dpi_config_t
MIPI DSI controller DPI interface configuration.

typedef struct *_dsi_dphy_config* dsi_dphy_config_t
MIPI DSI D-PHY configuration.

typedef enum *_dsi_tx_data_type* dsi_tx_data_type_t
DSI TX data type.

typedef enum *_dsi_rx_data_type* dsi_rx_data_type_t
DSI RX data type.

typedef struct *_dsi_transfer* dsi_transfer_t
Structure for the data transfer.

typedef struct *_dsi_handle* dsi_handle_t
MIPI DSI transfer handle.

typedef void (*dsi_callback_t)(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle, *status_t* status, void *userData)
MIPI DSI callback for finished transfer.

When transfer finished, one of these status values will be passed to the user:

- kStatus_Success Data transfer finished with no error.
- kStatus_Timeout Transfer failed because of timeout.
- kStatus_DSI_RxDataError RX data error, user could use DSI_GetRxErrorStatus to check the error details.
- kStatus_DSI_ErrorReportReceived Error Report packet received, user could use DSI_GetAndClearHostStatus to check the error report status.
- kStatus_Fail Transfer failed for other reasons.

FSL_DSI_TX_MAX_PAYLOAD_BYTE

FSL_DSI_RX_MAX_PAYLOAD_BYTE

struct _dsi_config
#include <fsl_mipi_dsi.h> MIPI DSI controller configuration.

Public Members

uint8_t numLanes
Number of lanes.

bool enableNonContinuousHsClk
: In enabled, the high speed clock will enter low power mode between transmissions.

bool enableTxUlps
: Enable the TX ULPS.

bool autoInsertEoTp
: Insert an EoTp short package when switching from HS to LP.

uint8_t numExtraEoTp
: How many extra EoTp to send after the end of a packet.

uint32_t htxTo_ByteClk
: HS TX timeout count (HTX_TO) in byte clock.

uint32_t lrxHostTo_ByteClk
: LP RX host timeout count (LRX-H_TO) in byte clock.

uint32_t btaTo_ByteClk
: Bus turn around timeout count (TA_TO) in byte clock.

struct _dsi_dpi_config
: *#include <fsl_mipi_dsi.h>* MIPI DSI controller DPI interface configuration.

Public Members

uint16_t pixelPayloadSize
: Maximum number of pixels that should be sent as one DSI packet. Recommended that the line size (in pixels) is evenly divisible by this parameter.

dsi_dpi_color_coding_t dpiColorCoding
: DPI color coding.

dsi_dpi_pixel_packet_t pixelPacket
: Pixel packet format.

dsi_dpi_video_mode_t videoMode
: Video mode.

dsi_dpi_bllp_mode_t bllpMode
: Behavior in BLLP.

uint8_t polarityFlags
: OR'ed value of _dsi_dpi_polarity_flag controls signal polarity.

uint16_t hfp
: Horizontal front porch, in dpi pixel clock.

uint16_t hbp
: Horizontal back porch, in dpi pixel clock.

uint16_t hsw
: Horizontal sync width, in dpi pixel clock.

uint8_t vfp
: Number of lines in vertical front porch.

uint8_t vbp
: Number of lines in vertical back porch.

uint16_t panelHeight
: Line number in vertical active area.

uint8_t virtualChannel

Virtual channel.

struct _dsi_dphy_config

#include <fsl_mipi_dsi.h> MIPI DSI D-PHY configuration.

Public Members

uint32_t txHsBitClk_Hz

The generated HS TX bit clock in Hz.

uint8_t tClkPre_ByteClk

TLPX + TCLK-PREPARE + TCLK-ZERO + TCLK-PRE in byte clock. Set how long the controller will wait after enabling clock lane for HS before enabling data lanes for HS.

uint8_t tClkPost_ByteClk

TCLK-POST + T_CLK-TRAIL in byte clock. Set how long the controller will wait before putting clock lane into LP mode after data lanes detected in stop state.

uint8_t tHsExit_ByteClk

THS-EXIT in byte clock. Set how long the controller will wait after the clock lane has been put into LP mode before enabling clock lane for HS again.

uint32_t tWakeup_EscClk

Number of clk_esc clock periods to keep a clock or data lane in Mark-1 state after exiting ULPS.

uint8_t tHsPrepare_HalfEscClk

THS-PREPARE in clk_esc/2. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3, 4, 5.

uint8_t tClkPrepare_HalfEscClk

TCLK-PREPARE in clk_esc/2. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3.

uint8_t tHsZero_ByteClk

THS-ZERO in clk_byte. Set how long that controller drives data lane HS-0 state before transmit the Sync sequence. Available values are 6, 7, ..., 37.

uint8_t tClkZero_ByteClk

TCLK-ZERO in clk_byte. Set how long that controller drives clock lane HS-0 state before transmit the Sync sequence. Available values are 3, 4, ..., 66.

uint8_t tHsTrail_ByteClk

THS-TRAIL + 4*UI in clk_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

uint8_t tClkTrail_ByteClk

TCLK-TRAIL + 4*UI in clk_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

struct _dsi_transfer

#include <fsl_mipi_dsi.h> Structure for the data transfer.

Public Members

uint8_t virtualChannel

Virtual channel.

dsi_tx_data_type_t txDataType
TX data type.

uint8_t flags
Flags to control the transfer, see _dsi_transfer_flags.

const uint8_t *txData
The TX data buffer.

uint8_t *rxData
The RX data buffer.

uint16_t txDataSize
Size of the TX data.

uint16_t rxDataSize
Size of the RX data.

bool sendDcsCmd
If set to true, the DCS command is specified by dcsCmd, otherwise the DCS command is included in the txData.

uint8_t dcsCmd
The DCS command to send, only valid when sendDcsCmd is true.

struct __dsi__handle
#include <fsl_mipi_dsi.h> MIPI DSI transfer handle structure.

Public Members

volatile bool isBusy
MIPI DSI is busy with APB data transfer.

dsi_transfer_t xfer
Transfer information.

dsi_callback_t callback
DSI callback

void *userData
Callback parameter

2.58 MIPI_DSI: MIPI DSI Host Controller

2.59 MMU: Memory Map Unit

void MMU__Init(MMU_Type *base, const *mmu_config_t* *config)
Initialize MMU:

a. Enable MMU clock;
b. Define the basic unit that is used in the memory map;
c. Configure valid address range;
d. Fill LUT SRAM;
e. Enable MMU remap function.

Parameters

- base – MMU peripheral base address.
- config – Pointer to the user defined variable in type of mmu_config_t.

void MMU_Deinit(MMU_Type *base)

De-initialize MMU:

a. Disable MMU remap function;
b. Reset MMU logic;
c. Disable MMU clock.

Parameters

- base –

void MMU_GetDefaultConfig(*mmu_config_t* *config)

Get default configurations of MMU module.

```
config->unitSize = kMMU_UnitSize4KB;
config->pageMode = kMMU_1PageMode;
config->invalidAddrMode = kMMU_InvalidAddrRespondError;
config->baseAddr     = 0x0UL;
config->maxAddr      = 0x0UL;
config->operationToWrite = kMMU_LUTBurstWrite;
config->ptrLutWord       = NULL;
config->countOfWordToWrite = 0U;
config->startPointer = 0U;
```

Parameters

- config – Pointer to the user defined variable in type of mmu_config_t.

void MMU_SetupSimpleMap(MMU_Type *base, uint32_t virt, uint32_t phys)

Map one physical unit to the virtual unit.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned virtual address.
- phys – The unit-aligned virtual address.

void MMU_TearDownSimpleMap(MMU_Type *base, uint32_t virt)

Remove mapping for one unit of provided virtual address.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned virtual address.

void MMU_SetupRegionMap(MMU_Type *base, uint32_t virt, uint32_t phys, size_t regionSize)

Map a region of physical memory into the virtual address space.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned destination virtual address to map.
- phys – The unit-aligned source physical address to map.
- regionSize – Unit-aligned size of mapped memory region in bytes.

void MMU_TearDownRegionMap(MMU_Type *base, uint32_t virt, size_t regionSize)

Remove mapping for a provided virtual address region.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned virtual address.
- regionSize – The unit-aligned size of region in bytes.

void MMU_RemapRegion(MMU_Type *base, uint32_t oldVirt, uint32_t newVirt, size_t regionSize)

Remap old virtual region into new address.

Parameters

- base – MMU peripheral base address.
- oldVirt – The unit-aligned old virtual address.
- newVirt – The unit-aligned new virtual address.
- regionSize – The unit-aligned size of region in bytes.

void MMU_MoveRegion(MMU_Type *base, uint32_t oldVirt, uint32_t newVirt, uint32_t newPhys, size_t regionSize)

Physically move region of memory, with copy.

Maps a region of physical memory into the new virtual address space, and copy region of size from the old virtual address space. The new virtual memory region is mapped from physical memory starting newPhys of size regionSize.

Parameters

- base – MMU peripheral base address.
- oldVirt – Unit-aligned old virtual region address.
- newVirt – Unit-aligned new virtual region address.
- newPhys – Unit-aligned new physical region address which will map to new virtual region address.
- regionSize – Unit-aligned size of region in bytes.

void MMU_SetupArrayMap(MMU_Type *base, uint32_t *ptrVirts, uint32_t *ptrPhys, uint32_t count)

Map an array of physical memory address into an array of virtual memory address.

Parameters

- base – MMU peripheral base address.
- ptrVirts – Pointer to the array which contain unit-aligned virtual address.
- ptrPhys – Pointer to the array which contain unit-aligned physical address.
- count – Count of array elements.

void MMU_TearDownArrayMap(MMU_Type *base, uint32_t *ptrVirts, uint32_t count)

Remove mapping of an array of virtual address.

Parameters

- base – MMU peripheral base address.
- ptrVirts – Pointer to the array which contains unit-aligned virtual address.
- count – Count of array elements.

void MMU_MoveArray(MMU_Type *base, uint32_t *ptrOldVirts, uint32_t *ptrNewVirts, uint32_t *ptrNewPhys, uint32_t count)

Physical move array of memory, with copy.

Maps an array of physical memory into the new array of virtual address space(ptrNewVirts), and copy an unit size of data from old array of virtual address to new array of virtual address.

Parameters

- base – MMU peripheral base address.
- ptrOldVirts – Pointer to the array which contain old unit-aligned virtual address.
- ptrNewVirts – Pointer to the array which contain new unit-aligned virtual address.
- ptrNewPhys – Pointer to the array which contain new unit-aligned physical address.
- count – Count of array elements.

void MMU_GetPhyAddr(MMU_Type *base, uint32_t virt, uint32_t *ptrPtrs)

Get mapped physical address from input virtual address.

Parameters

- base – MMU peripheral base address.
- virt – The virtual address.
- ptrPtrs – **[out]** Pointer to the variable to store mapped physical address.

static inline void MMU_EnableMemoryMap(MMU_Type *base, bool enable)

Enable/disable memory map.

Parameters

- base – MMU peripheral base address.
- enable – Used to enable/disable memory map:
  - **true** Enable memory map, the addresses of the AHB transaction are remapped by looking up LUT SRAM.
  - **false** Disable memory map, the AHB transaction passes through MMU without address remapping.

static inline void MMU_UpdateUnitSize(MMU_Type *base, *mmu_unit_size_t* unitSize)

Update the basic size of the unit that is used in the memory map.

Parameters

- base – MMU peripheral base address.
- unitSize – Used to specify the basic size of unit in type of mmu_unit_size_t.

void MMU_DoSoftwareReset(MMU_Type *base)

Trigger a software reset to reset the MMU logic.

Parameters

- base – MMU peripheral base address.

static inline void MMU_UpdateValidAddressRange(MMU_Type *base, uint16_t baseAddr, uint16_t maxAddr)

Update the valid address range.

```
To set AHB address from 0x51000000UL to 0x511FFFFF are valid:
MMU_UpdateValidAddressRange(MMU0, 0x5100U, 0x511FU);
```

Note: Both baseAddr and maxAddr mean the upper 16 bits of a valid address.

Parameters

- base – MMU peripheral base address.
- baseAddr – Upper 16 bits of valid range's base address.
- maxAddr – Upper 16 bits of valid range's maximum address.

static inline void MMU_ChangePageOption(MMU_Type *base, *mmu_page_option_t* pageOpt)

Change page mode, the MMU module supports #kMMU_1PageMode and #kMMU_4PageMode.

Parameters

- base – MMU peripheral base address.
- pageMode – Page mode to use, please refer to mmu_page_option_t.

void MMU_UpdateLUTBlocking(MMU_Type *base, uint16_t startPointer, *mmu_lut_word_t* *ptrLutWord, uint16_t countOfWords, *mmu_lut_write_operation_t* writeOperation)

Use the blocking method to update LUT SRAM with user-defined words.

Note: This function only return when all words that #ptrLutWord refers are set into LUT SRAM.

Note: This function supports both single write and burst write described in RM, please note that burst write is only useful when to fill continuous value into continuous addresses.

Note: startPointer + countOfWords should not exceed the maximum size of LUT SRAM(8192).

```
Fill continuous values into continuous addresses (E.g: initialize LUT SRAM address 2 and 3 with value␣
↪0x12 and 0x13)
        mmu_lut_word_t    startLutWord = {
            .mappedValue = 0x12U;
            .valid = 1U;
        };
        MMU_UpdateLUTBlocking(MMU0, 2U, &startLutWord, 2U, kMMU_LUTBurstWrite);
```

```
Fill discontinuous values into continuous addresses(E.g: initialize LUT SRAM address 2 and 3 with␣
↪value 0x12 and
0x34) mmu_lut_word_t lutWordArray[2] = {
            {
                .mappedValue = 0x12U;
                .valid       = 1U;
            },
            {
                .mappedValue = 0x34U;
                .valid       = 1U;
            },
```

(continues on next page)

(continued from previous page)

```
        };
        MMU_UpdateLUTBlocking(MMU0, 2U, lutWordArray, 2U,kMMU_LUTSingleWrite);
```

```
Fill discontinuous values into discontinuous addresses(E.g: initialize LUT SRAM address 2 and 5 with␣
↪value 0x12 and
0x34)): mmu_lut_word_t addr2LutWord = { .mappedValue = 0x12U; .valid      = 1U;
        };
        MMU_UpdateLUTBlocking(MMU0, 2U, &addr2LutWord, 1U, kMMU_LUTSingleWrite);

        mmu_lut_word_t addr5LutWord = {
            .mappedValue = 0x34U;
            .valid      = 1U;
        };
        MMU_UpdateLUTBlocking(MMU0, 5U, &addr5LutWord, 1U, kMMU_LUTSingleWrite);
```

Parameters

- base – MMU peripheral base address.
- startPointer – Start pointer of LUT SRAM to fill new mapping words, range from 0 to 8191.
- ptrLutWord – Pointer to new lut words: if **writeOperation** is selected as **kMMU_LUTBurstWrite** this parameter should be a pointer to the variable which represents the first LUT word to set; if **writeOperation** is selected as **kMMU_LUTSingleWrite** this parameter can be set as a pointer to the array that contains all new LUT words to set or set as a pointer to one variable when only want to update one word into LUT's specific pointer.
- countOfWord – Count to new word to update into LUT SRAM, range from 1 to 8192.
- writeOperation – Specify the write operation, the meaning of #ptrLutWord is different for different write operations.

void MMU_ReadLUT(MMU_Type *base, uint16_t startPointer, *mmu_lut_word_t* *ptrWordReadBuffer, uint16_t count)

Read the specified number of LUT content from the specified pointer into the user-defined buffer.

Note: #startPointer + #count should not exceed maximum size of LUT SRAM(8192).

Parameters

- base – MMU peripheral base address.
- startPointer – Specify start address to read in LUT SRAM, range from 0 to 8191.
- ptrWordReadBuffer – Pointer of a buffer that record read out words.
- count – Specify number of continuous address in LUT SRAM to read, range from 1 to 8192.

void MMU_TriggerLUTVerificationBlocking(MMU_Type *base, uint16_t startPointer, *mmu_lut_word_t* goldenWordToCheck, uint16_t count)

Trigger the specified number of read operations from LUT SRAM, compare with the input golden word, the read address and golden data are increased automatically each time.

Note: This function only return when all read-verify operations are complete.

Parameters

- base – MMU peripheral base address.
- startPointer – The pointer of LUT SRAM which start to check, range from 0 to 8191.
- goldenWordToCheck – The first golden word to check, in following read-verify operation, the golden data will increase automatically.
- count – Specify number of continuous address in LUT SRAM to check, range from 1 to 8192.

static inline void MMU_LockLUT(MMU_Type *base)

Lock LUT SRAM to avoid an accidental update.

Note: If locked, writing to LUT SRAM will trigger kMMU_LockErrorFlag.

Parameters

- base – MMU peripheral base address.

static inline void MMU_UnlockLUT(MMU_Type *base)

Unlock LUT SRAM.

Note: To update LUT SRAM, please ensure LUT is unlocked.

Parameters

- base – MMU peripheral base address.

static inline uint32_t MMU_GetStatusFlags(MMU_Type *base)

Get status flags.

Parameters

- base – MMU peripheral base address.

Returns

All status flags asserted, should be the OR'ed value of mmu_status_flags_t.

static inline void MMU_ClearStatusFlags(MMU_Type *base, uint32_t mask)

Clear status flags.

Parameters

- base – MMU peripheral base address.
- mask – The mask of flags to clear, should be the OR'ed value of mmu_status_flags_t.

static inline uint16_t MMU_GetIPSReadErrorAddress(MMU_Type *base)

Get the LUT SRAM address when an error is detected for the IPS read operation (including IPS read parity error and read-verify mismatch error).

Parameters

- base – MMU peripheral base address.

Returns
The pointer in LUT SRAM, range from 0 to 8191.

void MMU_GetAHBErrorRecord(MMU_Type *base, *mmu_ahb_error_record_t* *ptrRecord)

Get a record of AHB information when an error happens.

Parameters

- base – MMU peripheral base address.
- ptrRecord – Pointer to user-defined mmu_ahb_error_record_t variable which used to store current AHB error information.

Returns
The record of AHB information when an error happens, please refer to mmu_ahb_error_record_t for details.

static inline bool MMU_CheckMMUBusy(MMU_Type *base)

Check if MMU is in the Busy state.

Note: There are some possible reasons to make MMU is busy state:

- Initialization sequence
- Burst-write operation
- Read-verify operation

Parameters

- base – MMU peripheral base address.

Return values

- false – MMU not in the busy state.
- true – MMU is in the busy state, wait for it is not busy before taking an action.

static inline void MMU_EnableInterrupts(MMU_Type *base, uint32_t mask)

Enabled interrupts.

Parameters

- base – MMU peripheral base address.
- mask – Mask of interrupts to enable, should be the OR'ed value of mmu_interrupt_enable_t.

static inline void MMU_DisableInterrupts(MMU_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – MMU peripheral base address.
- mask – Mask of interrupts to disable, should be the OR'ed value of mmu_interrupt_enable_t.

FSL_MMU_DRIVER_VERSION

mmu driver version 2.0.0.

enum _mmu_status_flags

The enumeration of MMU status flags,.

Values:

enumerator kMMU_BusyFlag
MMU is in Busy state, wait for it to clear before taking action.

enumerator kMMU_LUTInvalidFlag
Indicate whether the valid bit in the access address is 0, which means the access operation is invalid in LUT.

enumerator kMMU_AddressInvalidFlag
The AHB access address is out of the valid address range.

enumerator kMMU_ParityErrorFlag
A parity error is detected in the access address.

enumerator kMMU_LockErrorFlag
A lock error is detected when writing to LUT SRAM.

enumerator kMMU_IpsReadErrorFlag
Indicate whether an IPS read error is detected when read LUT SRAM, including IPS read parity error and read-verify mismatch error.

enumerator kMMU_AllFlags
All flags.

enum _mmu_interrupt_enable
The enumeration of MMU interrupts,.

Values:

enumerator kMMU_LUTInvalidInterruptEnable
Enable the interrupt for kMMU_LUTInvalidFlag.

enumerator kMMU_AddressInvalidInterruptEnable
Enable the interrupt for kMMU_AddressInvalidFlag.

enumerator kMMU_ParityErrorInterruptEnable
Enable the interrupt for kMMU_ParityErrorFlag.

enumerator kMMU_LockErrorInterruptEnable
Enable the interrupt for kMMU_LockErrorFlag.

enumerator kMMU_IpsReadErrorInterruptEnable
Enable the interrupt for kMMU_IpsReadErrorFlag.

enumerator kMMU_AllInterruptsEnable
Enable all interrupts.

enum _mmu_unit_size
The enumeration of unit size, available selections are 4KB, 8KB, 16KB, 32KB, and 64KB.

Values:

enumerator kMMU_UnitSize4KB
Specify the basic size of unit is 4KB.

enumerator kMMU_UnitSize8KB
Specify the basic size of unit is 8KB.

enumerator kMMU_UnitSize16KB
Specify the basic size of unit is 16KB.

enumerator kMMU_UnitSize32KB
Specify the basic size of unit is 32KB.

enumerator kMMU_UnitSize64KB

Specify the basic size of unit is 64KB.

enum _mmu_ahb_access_type

The enumeration of AHB access type, available selections are read access and write access.

Values:

enumerator kMMU_ReadAccess

The access type of the AHB transfer is read.

enumerator kMMU_WriteAccess

The access type of the AHB transfer is write.

enum _mmu_ahb_error_type

The enumeration of error type in case of AHB transfer, available selections are LUT invalid error, address invalid error, and parity error.

Values:

enumerator kMMU_LUTInvalidError

The error type information of the transfer is LUT invalid error.

enumerator kMMU_AddrInvalidError

The error type information of the transfer is address invalid error.

enumerator kMMU_ParityError

The error type information of the transfer is parity error.

enum _mmu_page_option

The MMU module supports two different page mode: 1 page mode and 4 pages mode.

Note: In 4 pages mode, each page contains 2K word. Different page can contain different mapping schema.

Values:

enumerator kMMU_PageOpt_1Page

MMU takes LUT SRAM as 1 page for address remapping.

enumerator kMMU_PageOpt_4Page_Low2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM low 2K words for address remapping.

enumerator kMMU_PageOpt_4Page_MidLow2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM mid-low 2K words for address remapping.

enumerator kMMU_PageOpt_4Page_MidHigh2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM mid-high 2K words for address remapping.

enumerator kMMU_PageOpt_4Page_High2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM mid-low 2K words for address remapping.

enum _mmu_lut_write_operation

The enumeration of LUT SRAM write operation, both burst-write and single-write are supported.

Values:

enumerator kMMU_LUTSingleWrite

In this operation, words to write in consequent LUT pointer may not consequent. It is possible to write discontinuous value into discontinuous address.

enumerator kMMU_LUTBurstWrite

In this operation, words to write in consequence LUT pointer must be consequent. write continuous value into continuous address.

enum _mmu_invalid_address_mode

The behavior when an invalid address is detected is programmable.

Values:

enumerator kMMU_InvalidAddrRespondError

MMU responds with a transfer error when a invalid address is detected.

enumerator kMMU_InvalidAddrAllowed

MMU allows transfer when an invalid address is detected.

typedef enum *_mmu_unit_size* mmu_unit_size_t

The enumeration of unit size, available selections are 4KB, 8KB, 16KB, 32KB, and 64KB.

typedef enum *_mmu_ahb_access_type* mmu_ahb_access_type_t

The enumeration of AHB access type, available selections are read access and write access.

typedef enum *_mmu_ahb_error_type* mmu_ahb_error_type_t

The enumeration of error type in case of AHB transfer, available selections are LUT invalid error, address invalid error, and parity error.

typedef enum *_mmu_page_option* mmu_page_option_t

The MMU module supports two different page mode: 1 page mode and 4 pages mode.

Note: In 4 pages mode, each page contains 2K word. Different page can contain different mapping schema.

---

typedef enum *_mmu_lut_write_operation* mmu_lut_write_operation_t

The enumeration of LUT SRAM write operation, both burst-write and single-write are supported.

typedef enum *_mmu_invalid_address_mode* mmu_invalid_address_mode_t

The behavior when an invalid address is detected is programmable.

typedef struct *_mmu_ahb_error_record* mmu_ahb_error_record_t

The record of AHB information when an error happens.

typedef struct *_mmu_lut_word* mmu_lut_word_t

The word in LUT SRAM, there are 8K words exist in LUT SRAM.

typedef struct *_mmu_config* mmu_config_t

The structure used to initialize the MMU instance includes unit size, page mode, address range, and LUT content.

mmu_unit_size_t g_selectedUnitSize

mmu_page_option_t g_selectedPageOpt

MMU_NUM_OF_UNIT_SIZE

MMU_PAGE_OPT_MASK

MMU_PAGE_OPT_SHIFT

MMU_PAGE_OPT(x)

struct _mmu_ahb_error_record

#include <fsl_mmu.h> The record of AHB information when an error happens.

Public Members

uint32_t masterId

Indicate the master ID of the transfer.

uint32_t protectionInfo

Indicate the protection information of the transfer.

mmu_ahb_access_type_t accessType

Indicate the access type of the transfer.

mmu_ahb_error_type_t errorType

Indicate the error type of the transfer.

uint32_t failAddr

Indicates the LUT address where an erro happens.

struct _mmu_lut_word

#include <fsl_mmu.h> The word in LUT SRAM, there are 8K words exist in LUT SRAM.

Public Members

uint16_t mappedValue

The mapped value to set, should be index of unit in valid address range not specific AHB address.

uint16_t valid

Specify whether the mapped value is valid: 1b1 means valid, 1b0 means invalid.

struct _mmu_config

#include <fsl_mmu.h> The structure used to initialize the MMU instance includes unit size, page mode, address range, and LUT content.

Public Members

mmu_unit_size_t unitSize

Specify the basic size of unit that is used in the memory map.

mmu_page_option_t pageOpt

Specify the behavior of page option.

mmu_invalid_address_mode_t invalidAddrMode

Specify MMU behavior when an invalid address is detected.

uint16_t baseAddr

Upper 16 bits of valid range's base address.

uint16_t maxAddr

Upper 16 bits of valid range's maximum address.

uint32_t *ptrPhyAddrArray

Pointer to the physical address array used for array mapping.

uint32_t *ptrVirtualAddrArray
Pointer to the virtual address array used for array mapping.

uint16_t countOfAddr
Count of array elements

bool enableMMP
Enable/disable memory map feature, once enabled all AHB access to corresponding address are remapped by LUT SRAM.

2.60 MRT: Multi-Rate Timer

void MRT_Init(MRT_Type *base, const *mrt_config_t* *config)
Ungates the MRT clock and configures the peripheral for basic operation.

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

Parameters

- base – Multi-Rate timer peripheral base address
- config – Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MODCFG reigster, param config is useless.

void MRT_Deinit(MRT_Type *base)
Gate the MRT clock.

Parameters

- base – Multi-Rate timer peripheral base address

static inline void MRT_GetDefaultConfig(*mrt_config_t* *config)
Fill in the MRT config struct with the default settings.

The default values are:

```
config->enableMultiTask = false;
```

Parameters

- config – Pointer to user's MRT config structure.

static inline void MRT_SetupChannelMode(MRT_Type *base, *mrt_chnl_t* channel, const *mrt_timer_mode_t* mode)
Sets up an MRT channel mode.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Channel that is being configured.
- mode – Timer mode to use for the channel.

static inline void MRT_EnableInterrupts(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)
Enables the MRT interrupt.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

- mask – The interrupts to enable. This is a logical OR of members of the enumeration mrt_interrupt_enable_t

static inline void MRT_DisableInterrupts(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Disables the selected MRT interrupt.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- mask – The interrupts to disable. This is a logical OR of members of the enumeration mrt_interrupt_enable_t

static inline uint32_t MRT_GetEnabledInterrupts(MRT_Type *base, *mrt_chnl_t* channel)

Gets the enabled MRT interrupts.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns

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

static inline uint32_t MRT_GetStatusFlags(MRT_Type *base, *mrt_chnl_t* channel)

Gets the MRT status flags.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns

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

static inline void MRT_ClearStatusFlags(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Clears the MRT status flags.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration mrt_status_flags_t

void MRT_UpdateTimerPeriod(MRT_Type *base, *mrt_chnl_t* channel, uint32_t count, bool immediateLoad)

Used to update the timer period in units of count.

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks

---

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

- count – Timer period in units of ticks
- immediateLoad – true: Load the new value immediately into the TIMER register; false: Load the new value at the end of current timer interval

static inline uint32_t MRT_GetCurrentTimerCount(MRT_Type *base, *mrt_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in fsl_common.h to convert ticks to usec or msec

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns

Current timer counting value in ticks

static inline void MRT_StartTimer(MRT_Type *base, *mrt_chnl_t* channel, uint32_t count)

Starts the timer counting.

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.
- count – Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

static inline void MRT_StopTimer(MRT_Type *base, *mrt_chnl_t* channel)

Stops the timer counting.

This function stops the timer from counting.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

static inline uint32_t MRT_GetIdleChannel(MRT_Type *base)

Find the available channel.

This function returns the lowest available channel number.

Parameters

- base – Multi-Rate timer peripheral base address

static inline void MRT_ReleaseChannel(MRT_Type *base, *mrt_chnl_t* channel)

Release the channel when the timer is using the multi-task mode.

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling

MRT_GetIdleChannel() for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

FSL_MRT_DRIVER_VERSION

enum _mrt_chnl

List of MRT channels.

Values:

enumerator kMRT_Channel_0

MRT channel number 0

enumerator kMRT_Channel_1

MRT channel number 1

enumerator kMRT_Channel_2

MRT channel number 2

enumerator kMRT_Channel_3

MRT channel number 3

enum _mrt_timer_mode

List of MRT timer modes.

Values:

enumerator kMRT_RepeatMode

Repeat Interrupt mode

enumerator kMRT_OneShotMode

One-shot Interrupt mode

enumerator kMRT_OneShotStallMode

One-shot stall mode

enum _mrt_interrupt_enable

List of MRT interrupts.

Values:

enumerator kMRT_TimerInterruptEnable

Timer interrupt enable

enum _mrt_status_flags

List of MRT status flags.

Values:

enumerator kMRT_TimerInterruptFlag

Timer interrupt flag

enumerator kMRT_TimerRunFlag

Indicates state of the timer

typedef enum *_mrt_chnl* mrt_chnl_t

List of MRT channels.

typedef enum *_mrt_timer_mode* mrt_timer_mode_t

List of MRT timer modes.

typedef enum *_mrt_interrupt_enable* mrt_interrupt_enable_t

List of MRT interrupts.

typedef enum *_mrt_status_flags* mrt_status_flags_t

List of MRT status flags.

typedef struct *_mrt_config* mrt_config_t

MRT configuration structure.

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

The config struct can be made const so it resides in flash

struct _mrt_config

#include <fsl_mrt.h> MRT configuration structure.

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

The config struct can be made const so it resides in flash

Public Members

bool enableMultiTask

true: Timers run in multi-task mode; false: Timers run in hardware status mode

2.61 MU: Messaging Unit

uint32_t MU_GetInstance(MU_Type *base)

Get the MU instance index.

Parameters

- base – MU peripheral base address.

Returns

MU instance index.

void MU_Init(MU_Type *base)

Initializes the MU module.

This function enables the MU clock only.

Parameters

- base – MU peripheral base address.

void MU_Deinit(MU_Type *base)

De-initializes the MU module.

This function disables the MU clock only.

Parameters

- base – MU peripheral base address.

static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)

Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { }  Wait for TX0 register empty.
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL);  Write message to the TX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – TX register index, see mu_msg_reg_index_t.
- msg – Message to send.

status_t MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)

Blocks to send a message.

This function waits until the TX register is empty and sends the message. If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and returns kStatus_Timeout.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see mu_msg_reg_index_t.
- msg – Message to send.

Return values

- kStatus_Success – Message sent successfully.
- kStatus_Timeout – Timeout occurred while waiting for TX register to be empty.

Returns

status_t

static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
}  Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0);  Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.

Returns

The received message.

status_t MU__ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

This function provides the same blocking behavior as MU_ReceiveMsg() but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

Note: Both MU_ReceiveMsg() and MU_ReceiveMsgTimeout() are blocking functions. The difference is that this function includes timeout protection while MU_ReceiveMsg() waits indefinitely.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.
- readValue – Pointer to store the received message.

Return values

- kStatus_Success – Message received successfully.
- kStatus_InvalidArgument – Invalid readValue pointer.
- kStatus_Timeout – Timeout occurred while waiting for RX register to be full.

Returns

status_t

uint32_t MU__ReceiveMsg(MU_Type *base, uint32_t regIndex)

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

Note: Both MU_ReceiveMsg() and MU_ReceiveMsgTimeout() are blocking functions. The difference is that MU_ReceiveMsgTimeout() includes timeout protection while this function waits indefinitely.

Warning: This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use MU_ReceiveMsgTimeout() instead.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see mu_msg_reg_index_t.

Returns

The received message.

static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU_FlagsUpdatingFlag is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
}  Wait for previous MU flags updating.

MU_SetFlagsNonBlocking(base, 0U);  Set the mU flags.
```

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

status_t MU_SetFlags(MU_Type *base, uint32_t flags)

brief Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

return status_t retval kStatus_Success Flags were set successfully. retval kStatus_Timeout Timeout occurred while waiting for flags to update.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

static inline uint32_t MU_GetFlags(MU_Type *base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

uint32_t MU_GetStatusFlags(MU_Type *base)

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See _mu_status_flags.

```
uint32_t flags;
flags = MU_GetStatusFlags(base);  Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
```

(continues on next page)

(continued from previous page)

```
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

If there are more than 4 general purpose interrupts, use MU_GetGeneralPurposeStatusFlags.

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU status flags, see _mu_status_flags.

static inline uint32_t MU_GetInterruptsPending(MU_Type *base)

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. kMU_Tx0EmptyFlag kMU_Tx1EmptyFlag kMU_Tx2EmptyFlag kMU_Tx3EmptyFlag kMU_Rx0FullFlag kMU_Rx1FullFlag kMU_Rx2FullFlag kMU_Rx3FullFlag kMU_GenInt0Flag kMU_GenInt1Flag kMU_GenInt2Flag kMU_GenInt3Flag

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t flags)

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See _mu_status_flags.

```
Clear general interrupt 0 and general interrupt 1 pending flags.
MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);
```

If there are more than 4 general purpose interrupts, use MU_ClearGeneralPurposeStatusFlags.

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU status flags. See _mu_status_flags. Only the following flags can be cleared by software (if applicable for particular device), other flags are cleared by hardware:
  - kMU_GenInt0Flag
  - kMU_GenInt1Flag
  - kMU_GenInt2Flag
  - kMU_GenInt3Flag
  - kMU_MuResetInterruptFlag
  - kMU_OtherSideEnterRunInterruptFlag

- kMU_OtherSideEnterHaltInterruptFlag
- kMU_OtherSideEnterWaitInterruptFlag
- kMU_OtherSideEnterStopInterruptFlag
- kMU_OtherSideEnterPowerDownInterruptFlag
- kMU_ResetAssertInterruptFlag
- kMU_HardwareResetInterruptFlag

static inline void MU_EnableInterrupts(MU_Type *base, uint32_t interrupts)

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See _mu_interrupt_enable.

```
  Enable general interrupt 0 and TX0 empty interrupt.
MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use MU_EnableGeneralPurposeInterrupts.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See _mu_interrupt_enable.

static inline void MU_DisableInterrupts(MU_Type *base, uint32_t interrupts)

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See _mu_interrupt_enable.

```
  Disable general interrupt 0 and TX0 empty interrupt.
MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use MU_DisableGeneralPurposeInterrupts.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See _mu_interrupt_enable.

status_t MU_TriggerInterrupts(MU_Type *base, uint32_t interrupts)

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See _mu_interrupt_trigger. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

If there are more than 4 general purpose interrupts, use MU_TriggerGeneralPurposeInterrupts.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See _mu_interrupt_trigger.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

static inline void MU_EnableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Enables the MU general purpose interrupts.

This function enables the MU general purpose interrupts. The interrupts to enable should be passed in as bit mask of mu_general_purpose_interrupt_t. The function MU_EnableInterrupts only support general interrupt 0~3, this function supports all general interrupts.

For example, to enable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts, see mu_general_purpose_interrupt_t.

static inline void MU_DisableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Disables the MU general purpose interrupts.

This function disables the MU general purpose interrupts. The interrupts to disable should be passed in as bit mask of mu_general_purpose_interrupt_t. The function MU_DisableInterrupts only support general interrupt 0~3, this function supports all general interrupts.

For example, to disable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts. see mu_general_purpose_interrupt_t.

static inline uint32_t MU_GetGeneralPurposeStatusFlags(MU_Type *base)

Gets the MU general purpose interrupt status flags.

This function returns the bit mask of the MU general purpose interrupt status flags. MU_GetStatusFlags can only get general purpose interrupt status 0~3, this function can get all general purpose interrupts status.

This example shows to check whether general purpose interrupt 0 and 3 happened.

```
uint32_t flags;
flags = MU_GetGeneralPurposeStatusFlags(base);
if (kMU_GeneralPurposeInterrupt0 & flags)
{
}
if (kMU_GeneralPurposeInterrupt3 & flags)
```

(continues on next page)

(continued from previous page)

```
{
}
```

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU general purpose interrupt status flags.

static inline void MU_ClearGeneralPurposeStatusFlags(MU_Type *base, uint32_t flags)

Clear the MU general purpose interrupt status flags.

This function clears the specific MU general purpose interrupt status flags. The flags to clear should be passed in as bit mask. mu_general_purpose_interrupt_t_mu_status_flags.

Example to clear general purpose interrupt 0 and general interrupt 1 pending flags.

```
MU_ClearGeneralPurposeStatusFlags(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt1);
```

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU general purpose interrupt status flags. See mu_general_purpose_interrupt_t.

static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)

Return the RX status flags in reverse numerical order.

This function return the RX status flags in reverse order. Note: RFn bits of SR[3-0](mu status register) are mapped in ascending numerical order: RF0 -> SR[0] RF1 -> SR[1] RF2 -> SR[2] RF3 -> SR[3] This function will return these bits in reverse numerical order(RF3->RF1) to comply with MU_GetRxStatusFlags() of mu driver. See MU_GetRxStatusFlags() from drivers/mu/fsl_mu.h

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns

MU RX status flags in reverse order

status_t MU_TriggerGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Triggers general purpose interrupts to the other core.

This function triggers the specific general purpose interrupts to the other core. The interrupts to trigger are passed in as bit mask. See mu_general_purpose_interrupt_t. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
status_t status;
status = MU_TriggerGeneralPurposeInterrupts(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt2);

if (kStatus_Success != status)
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
```

(continues on next page)

(continued from previous page)

```
    has not been processed by the other core.
}
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See mu_general_purpose_interrupt_t.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

void MU_BootOtherCore(MU_Type *base, mu_core_boot_mode_t mode)

Boots the other core.

This function boots the other core with a boot configuration.

Parameters

- base – MU peripheral base address.
- mode – The other core boot mode.

void MU_HoldOtherCoreReset(MU_Type *base)

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

Parameters

- base – MU peripheral base address.

static inline *status_t* MU_ResetBothSides(MU_Type *base)

Resets the MU for both A side and B side.

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations if waiting for the other side to come out of reset takes too long.

Note: For some platforms, only MU side A could use this function, check reference manual for details.

Parameters

- base – MU peripheral base address.

Return values

- kStatus_Success – The MU was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other side to come out of reset.

Returns

status_t

status_t MU_HardwareResetOtherCore(MU_Type *base, bool waitReset, bool holdReset, mu_core_boot_mode_t bootMode)

Hardware reset the other core.

This function resets the other core, the other core could mask the hardware reset by calling MU_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU_BootOtherCore to control the other core reset workflow.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout if waiting for the other core to enter or exit reset takes too long.

Example 1: Reset the other core, and no hold reset

```
MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later. Here the other core enters reset, and the reset is hold

```
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
```

Current core boot the other core when necessary.

```
MU_BootOtherCore(MU_A, bootMode);
```

Note: The feature waitReset, holdReset, and bootMode might be not supported for some platforms. waitReset is only available for platforms that FSL_FEATURE_MU_NO_CORE_STATUS not defined as 1 and FSL_FEATURE_MU_HAS_RESET_ASSERT_INT not defined as 0. holdReset is only available for platforms that FSL_FEATURE_MU_HAS_RSTH not defined as 0. bootMode is only available for platforms that FSL_FEATURE_MU_HAS_BOOT not defined as 0.

Parameters

- base – MU peripheral base address.
- waitReset – Wait the other core enters reset. Only work when there is CSSR0[RAIP].
  - true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
  - false: Don't wait the reset.
- holdReset – Hold the other core reset or not. Only work when there is CCR0[RSTH].
  - true: Hold the other core in reset, this function returns directly when the other core enters reset.
  - false: Don't hold the other core in reset, this function waits until the other core out of reset.
- bootMode – Boot mode of the other core, if holdReset is true, this parameter is useless.

Return values

- kStatus_Success – The other core was reset successfully.

- kStatus_Timeout – Timeout occurred while waiting for the other core to enter or exit reset.

Returns
status_t

FSL_MU_DRIVER_VERSION
MU driver version.

enum _mu_status_flags
MU status flags.

Values:

enumerator kMU_Tx0EmptyFlag
TX0 empty.

enumerator kMU_Tx1EmptyFlag
TX1 empty.

enumerator kMU_Tx2EmptyFlag
TX2 empty.

enumerator kMU_Tx3EmptyFlag
TX3 empty.

enumerator kMU_Rx0FullFlag
RX0 full.

enumerator kMU_Rx1FullFlag
RX1 full.

enumerator kMU_Rx2FullFlag
RX2 full.

enumerator kMU_Rx3FullFlag
RX3 full.

enumerator kMU_GenInt0Flag
General purpose interrupt 0 pending.

enumerator kMU_GenInt1Flag
General purpose interrupt 1 pending.

enumerator kMU_GenInt2Flag
General purpose interrupt 2 pending.

enumerator kMU_GenInt3Flag
General purpose interrupt 3 pending.

enumerator kMU_RxFullPendingFlag
Any RX full flag is pending.

enumerator kMU_TxEmptyPendingFlag
Any TX empty flag is pending.

enumerator kMU_GenIntPendingFlag
Any general interrupt flag is pending.

enumerator kMU_EventPendingFlag
MU event pending.

enumerator kMU_FlagsUpdatingFlag
MU flags update is on-going.

enumerator kMU_MuInResetFlag
MU of any side is in reset.

enumerator kMU_MuResetInterruptFlag
The other side initializes MU reset.

enum _mu_interrupt_enable
MU interrupt source to enable.

Values:

enumerator kMU_Tx0EmptyInterruptEnable
TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable
TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable
TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable
TX3 empty.

enumerator kMU_Rx0FullInterruptEnable
RX0 full.

enumerator kMU_Rx1FullInterruptEnable
RX1 full.

enumerator kMU_Rx2FullInterruptEnable
RX2 full.

enumerator kMU_Rx3FullInterruptEnable
RX3 full.

enumerator kMU_GenInt0InterruptEnable
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable
General purpose interrupt 3.

enumerator kMU_MuResetInterruptEnable
The other side initializes MU reset.

enum _mu_interrupt_trigger
MU interrupt that could be triggered to the other core.

Values:

enumerator kMU_GenInt0InterruptTrigger
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptTrigger
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptTrigger
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptTrigger
General purpose interrupt 3.

enum _mu_msg_reg_index
MU message register index.

Values:

enumerator kMU_MsgReg0
Message register 0.

enumerator kMU_MsgReg1
Message register 1.

enumerator kMU_MsgReg2
Message register 2.

enumerator kMU_MsgReg3
Message register 3.

enum _mu_general_purpose_interrupt
MU general purpose interrupts.

Values:

enumerator kMU_GeneralPurposeInterrupt0
General purpose interrupt 0

enumerator kMU_GeneralPurposeInterrupt1
General purpose interrupt 1

enumerator kMU_GeneralPurposeInterrupt2
General purpose interrupt 2

enumerator kMU_GeneralPurposeInterrupt3
General purpose interrupt 3

typedef enum *_mu_msg_reg_index* mu_msg_reg_index_t
MU message register index.

typedef enum *_mu_general_purpose_interrupt* mu_general_purpose_interrupt_t
MU general purpose interrupts.

MU_CORE_INTR(intr)

MU_MISC_INTR(intr)

MU_TX_INTR(intr)

MU_RX_INTR(intr)

MU_GI_INTR(intr)

MU_GET_CORE_INTR(intrs)

MU_GET_TX_INTR(intrs)

MU_GET_RX_INTR(intrs)

MU_GET_GI_INTR(intrs)

MU_CORE_FLAG(flag)

MU_STAT_FLAG(flag)

MU_TX_FLAG(flag)

MU_RX_FLAG(flag)

MU_GI_FLAG(flag)

MU_GET_CORE_FLAG(flags)

MU_GET_STAT_FLAG(flags)

MU_GET_TX_FLAG(flags)

MU_GET_RX_FLAG(flags)

MU_GET_GI_FLAG(flags)

MU1_BUSY_POLL_COUNT

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

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 a core becomes unresponsive.

2.62 This type defines status return values used by NBOOT functions that are not easily

kStatus_NBOOT_Success

Operation completed successfully.

kStatus_NBOOT_Fail

Operation failed.

kStatus_NBOOT_InvalidArgument

Invalid argument passed to the function.

kStatus_NBOOT_RequestTimeout

Operation timed out.

kStatus_NBOOT_KeyNotLoaded

The requested key is not loaded.

kStatus_NBOOT_AuthFail

Authentication failed.

kStatus_NBOOT_OperationNotAvaialable

Operation not available on this HW.

kStatus_NBOOT_KeyNotAvailable

Key is not avaialble.

kStatus_NBOOT_IvCounterOverflow

Overflow of IV counter (PRINCE/IPED).

kStatus_NBOOT_SelftestFail
: FIPS self-test failure.

kStatus_NBOOT_InvalidDataFormat
: Invalid data format for example antipole

kStatus_NBOOT_IskCertUserDataTooBig
: Size of User data in ISK certificate is greater than 96 bytes

kStatus_NBOOT_IskCertSignatureOffsetTooSmall
: Signature offset in ISK certificate is smaller than expected

kStatus_NBOOT_MemcpyFail
: Unexpected error detected during nboot_memcpy()

2.63 OSTIMER: OS Event Timer Driver

void OSTIMER_Init(OSTIMER_Type *base)
: Initializes an OSTIMER by turning its bus clock on.

void OSTIMER_Deinit(OSTIMER_Type *base)
: Deinitializes a OSTIMER instance.

  This function shuts down OSTIMER bus clock

  Parameters

  - base – OSTIMER peripheral base address.

uint64_t OSTIMER_GrayToDecimal(uint64_t gray)
: Translate the value from gray-code to decimal.

  Parameters

  - gray – The gray value input.

  Returns
  The decimal value.

static inline uint64_t OSTIMER_DecimalToGray(uint64_t dec)
: Translate the value from decimal to gray-code.

  Parameters

  - dec – The decimal value.

  Returns
  The gray code of the input value.

uint32_t OSTIMER_GetStatusFlags(OSTIMER_Type *base)
: Get OSTIMER status Flags.

  This returns the status flag. Currently, only match interrupt flag can be got.

  Parameters

  - base – OSTIMER peripheral base address.

  Returns
  status register value

void OSTIMER_ClearStatusFlags(OSTIMER_Type *base, uint32_t mask)

Clear Status Interrupt Flags.

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

- base – OSTIMER peripheral base address.
- mask – Clear bit mask.

Returns

none

status_t OSTIMER_SetMatchRawValue(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match raw value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format may be gray-code, if so, please using OSTIMER_SetMatchValue().

Parameters

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value may be gray-code format)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success – - Set match raw value and enable interrupt Successfully.

status_t OSTIMER_SetMatchValue(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

Parameters

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code in API if the IP counter is gray encoded.)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success – - Set match raw value and enable interrupt Successfully.

static inline void OSTIMER_SetMatchRegister(OSTIMER_Type *base, uint64_t value)

Set value to OSTIMER MATCH register directly.

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER_DecimalToGray could convert decimal value to gray code.

Parameters

- base – OSTIMER peripheral base address.
- value – OSTIMER timer match value (Value is gray-code format).

static inline uint64_t OSTIMER_GetMatchRegister(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get the match value from OSTIMER. The value of timer match is gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register, data format is gray code.

static inline uint64_t OSTIMER_GetMatchValue(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get a match value from OSTIMER.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register.

static inline void OSTIMER_EnableMatchInterrupt(OSTIMER_Type *base)

Enable the OSTIMER counter match interrupt.

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline void OSTIMER_DisableMatchInterrupt(OSTIMER_Type *base)

Disable the OSTIMER counter match interrupt.

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline uint64_t OSTIMER_GetCurrentTimerRawValue(OSTIMER_Type *base)

Get current timer raw count value from OSTIMER.

This function will get the timer count value from OS timer register. The raw value of timer count may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, may be gray code format.

uint64_t OSTIMER_GetCurrentTimerValue(OSTIMER_Type *base)

Get current timer count value from OSTIMER.

This function will get a decimal timer count value. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formated to decimal value.

static inline uint64_t OSTIMER_GetCaptureRawValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a captured value from OSTIMER. The Raw value of timer capture may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of capture register, data format may be gray code.

uint64_t OSTIMER_GetCaptureValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a capture decimal-value from OSTIMER. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of capture register, data format is decimal.

void OSTIMER_HandleIRQ(OSTIMER_Type *base, *ostimer_callback_t* cb)

OS timer interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – OS timer peripheral base address.
- cb – callback scheduled for this instance of OS timer

Returns

none

FSL_OSTIMER_DRIVER_VERSION

OSTIMER driver version.

enum _ostimer_flags

OSTIMER status flags.

Values:

enumerator kOSTIMER_MatchInterruptFlag

Match interrupt flag bit, sets if the match value was reached.

typedef void (*ostimer_callback_t)(void)

ostimer callback function.

2.64 PDM: Microphone Interface

2.65 PDM Driver

void PDM_Init(PDM_Type *base, const *pdm_config_t* *config)

Initializes the PDM peripheral.

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by PDM_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

Parameters

- base – PDM base pointer
- config – PDM configuration structure.

void PDM_Deinit(PDM_Type *base)

De-initializes the PDM peripheral.

This API gates the PDM clock. The PDM module can't operate unless PDM_Init is called to enable the clock.

Parameters

- base – PDM base pointer

static inline void PDM_Reset(PDM_Type *base)

Resets the PDM module.

Parameters

- base – PDM base pointer

static inline void PDM_Enable(PDM_Type *base, bool enable)

Enables/disables PDM interface.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled, false means PDM interface is disabled.

static inline void PDM_EnableDebugMode(PDM_Type *base, bool enable)

Enables/disables debug mode for PDM. The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface enter debug mode, false means PDM interface in normal mode.

static inline void PDM_EnableInDebugMode(PDM_Type *base, bool enable)

Enables/disables PDM interface in debug mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode.

static inline void PDM_EnterLowLeakageMode(PDM_Type *base, bool enable)

Enables/disables PDM interface disable/Low Leakage mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode.

static inline void PDM_EnableChannel(PDM_Type *base, uint8_t channel, bool enable)

Enables/disables the PDM channel.

Parameters

- base – PDM base pointer
- channel – PDM channel number need to enable or disable.
- enable – True means enable PDM channel, false means disable.

void PDM_SetChannelConfig(PDM_Type *base, uint32_t channel, const *pdm_channel_config_t* *config)

PDM one channel configurations.

Parameters

- base – PDM base pointer
- config – PDM channel configurations.
- channel – channel number. after completing the current frame in debug mode.

status_t PDM_SetSampleRateConfig(PDM_Type *base, uint32_t sourceClock_HZ, uint32_t sampleRate_HZ)

PDM set sample rate.

Note: This function is depend on the configuration of the PDM and PDM channel, so the correct call sequence is

```
PDM_Init(base, pdmConfig)
PDM_SetChannelConfig(base, channel, &channelConfig)
PDM_SetSampleRateConfig(base, source, sampleRate)
```

Parameters

- base – PDM base pointer
- sourceClock_HZ – PDM source clock frequency.
- sampleRate_HZ – PDM sample rate.

status_t PDM_SetSampleRate(PDM_Type *base, uint32_t enableChannelMask, *pdm_df_quality_mode_t* qualityMode, uint8_t osr, uint32_t clkDiv)

PDM set sample rate.

Deprecated:

Do not use this function. It has been superceded by PDM_SetSampleRateConfig

Parameters

- base – PDM base pointer

- enableChannelMask – PDM channel enable mask.
- qualityMode – quality mode.
- osr – cic oversample rate
- clkDiv – clock divider

uint32_t PDM_GetInstance(PDM_Type *base)

Get the instance number for PDM.

Parameters

- base – PDM base pointer.

static inline uint32_t PDM_GetStatus(PDM_Type *base)

Gets the PDM internal status flag. Use the Status Mask in _pdm_internal_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

PDM status flag value.

static inline uint32_t PDM_GetFifoStatus(PDM_Type *base)

Gets the PDM FIFO status flag. Use the Status Mask in _pdm_fifo_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

FIFO status.

static inline uint32_t PDM_GetRangeStatus(PDM_Type *base)

Gets the PDM Range status flag. Use the Status Mask in _pdm_range_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

output status.

static inline void PDM_ClearStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status between kPDM_StatusFrequencyLow and kPDM_StatusCh7FifoDataAvaliable.

static inline void PDM_ClearFIFOStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask.It can be a combination of the status in _pdm_fifo_status.

static inline void PDM_ClearRangeStatus(PDM_Type *base, uint32_t mask)

Clears the PDM range status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in _pdm_range_status.

void PDM_EnableInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_ErrorInterruptEnable
  - kPDM_FIFOInterruptEnable

static inline void PDM_DisableInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_ErrorInterruptEnable
  - kPDM_FIFOInterruptEnable

static inline void PDM_EnableDMA(PDM_Type *base, bool enable)

Enables/disables the PDM DMA requests.

Parameters

- base – PDM base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t PDM_GetDataRegisterAddress(PDM_Type *base, uint32_t channel)

Gets the PDM data register address.

This API is used to provide a transfer address for the PDM DMA transfer configuration.

Parameters

- base – PDM base pointer.
- channel – Which data channel used.

Returns

data register address.

void PDM_ReadFifo(PDM_Type *base, uint32_t startChannel, uint32_t channelNums, void *buffer, size_t size, uint32_t dataWidth)

PDM read fifo.

Note: : This function support 16 bit only for IP version that only supports 16bit.

Parameters

- base – PDM base pointer.
- startChannel – start channel number.
- channelNums – total enabled channelnums.
- buffer – received buffer address.
- size – number of samples to read.
- dataWidth – sample width.

void PDM_SetChannelGain(PDM_Type *base, uint32_t channel, *pdm_df_output_gain_t* gain)

Set the PDM channel gain.

Please note for different quality mode, the valid gain value is different, reference RM for detail.

Parameters

- base – PDM base pointer.
- channel – PDM channel index.
- gain – channel gain, the register gain value range is 0 - 15.

void PDM_SetHwvadConfig(PDM_Type *base, const *pdm_hwvad_config_t* *config)

Configure voice activity detector.

Parameters

- base – PDM base pointer
- config – Voice activity detector configure structure pointer .

static inline void PDM_ForceHwvadOutputDisable(PDM_Type *base, bool enable)

PDM hwvad force output disable.

Parameters

- base – PDM base pointer
- enable – true is output force disable, false is output not force.

static inline void PDM_ResetHwvad(PDM_Type *base)

PDM hwvad reset. It will reset VADNDATA register and will clean all internal buffers, should be called when the PDM isn't running.

Parameters

- base – PDM base pointer

static inline void PDM_EnableHwvad(PDM_Type *base, bool enable)

Enable/Disable Voice activity detector. Should be called when the PDM isn't running.

Parameters

- base – PDM base pointer.
- enable – True means enable voice activity detector, false means disable.

static inline void PDM_EnableHwvadInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM Voice Detector interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_HWVADErrorInterruptEnable

– kPDM_HWVADInterruptEnable

static inline void PDM_DisableHwvadInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM Voice Detector interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_HWVADErrorInterruptEnable
  - kPDM_HWVADInterruptEnable

static inline void PDM_ClearHwvadInterruptStatusFlags(PDM_Type *base, uint32_t mask)

Clears the PDM voice activity detector status flags.

Parameters

- base – PDM base pointer
- mask – State mask,reference _pdm_hwvad_int_status.

static inline uint32_t PDM_GetHwvadInterruptStatusFlags(PDM_Type *base)

Clears the PDM voice activity detector status flags.

Parameters

- base – PDM base pointer

Returns

status, reference _pdm_hwvad_int_status

static inline uint32_t PDM_GetHwvadInitialFlag(PDM_Type *base)

Get the PDM voice activity detector initial flags.

Parameters

- base – PDM base pointer

Returns

initial flag.

static inline void PDM_EnableHwvadSignalFilter(PDM_Type *base, bool enable)

Enables/disables voice activity detector signal filter.

Parameters

- base – PDM base pointer
- enable – True means enable signal filter, false means disable.

void PDM_SetHwvadSignalFilterConfig(PDM_Type *base, bool enableMaxBlock, uint32_t signalGain)

Configure voice activity detector signal filter.

Parameters

- base – PDM base pointer
- enableMaxBlock – If signal maximum block enabled.
- signalGain – Gain value for the signal energy.

void PDM_SetHwvadNoiseFilterConfig(PDM_Type *base, const *pdm_hwvad_noise_filter_t* *config)

Configure voice activity detector noise filter.

Parameters

- base – PDM base pointer
- config – Voice activity detector noise filter configure structure pointer .

static inline void PDM_EnableHwvadZeroCrossDetector(PDM_Type *base, bool enable)

Enables/disables voice activity detector zero cross detector.

Parameters

- base – PDM base pointer
- enable – True means enable zero cross detector, false means disable.

void PDM_SetHwvadZeroCrossDetectorConfig(PDM_Type *base, const *pdm_hwvad_zero_cross_detector_t* *config)

Configure voice activity detector zero cross detector.

Parameters

- base – PDM base pointer
- config – Voice activity detector zero cross detector configure structure pointer .

static inline uint16_t PDM_GetNoiseData(PDM_Type *base)

Reads noise data.

Parameters

- base – PDM base pointer.

Returns

Data in PDM noise data register.

static inline void PDM_SetHwvadInternalFilterStatus(PDM_Type *base, *pdm_hwvad_filter_status_t* status)

set hwvad internal filter status . Note: filter initial status should be asserted for two more cycles, then set it to normal operation.

Parameters

- base – PDM base pointer.
- status – internal filter status.

void PDM_SetHwvadInEnvelopeBasedMode(PDM_Type *base, const *pdm_hwvad_config_t* *hwvadConfig, const *pdm_hwvad_noise_filter_t* *noiseConfig, const *pdm_hwvad_zero_cross_detector_t* *zcdConfig, uint32_t signalGain)

set HWVAD in envelope based mode . Recommand configurations,

```
  static const pdm_hwvad_config_t hwvadConfig = {
    .channel          = 0,
    .initializeTime   = 10U,
    .cicOverSampleRate = 0U,
    .inputGain        = 0U,
    .frameTime        = 10U,
    .cutOffFreq       = kPDM_HwvadHpfBypassed,
    .enableFrameEnergy = false,
    .enablePreFilter  = true,
};

  static const pdm_hwvad_noise_filter_t noiseFilterConfig = {
    .enableAutoNoiseFilter = false,
```

(continues on next page)

(continued from previous page)

```
    .enableNoiseMin        = true,
    .enableNoiseDecimation = true,
    .noiseFilterAdjustment = 0U,
    .noiseGain             = 7U,
    .enableNoiseDetectOR   = true,
};
```

Parameters

- base – PDM base pointer.
- hwvadConfig – internal filter status.
- noiseConfig – Voice activity detector noise filter configure structure pointer.
- zcdConfig – Voice activity detector zero cross detector configure structure pointer .
- signalGain – signal gain value.

void PDM_SetHwvadInEnergyBasedMode(PDM_Type *base, const *pdm_hwvad_config_t* *hwvadConfig, const *pdm_hwvad_noise_filter_t* *noiseConfig, const *pdm_hwvad_zero_cross_detector_t* *zcdConfig, uint32_t signalGain)

brief set HWVAD in energy based mode . Recommand configurations, code static const pdm_hwvad_config_t hwvadConfig = { .channel = 0, .initializeTime = 10U, .cicOverSampleRate = 0U, .inputGain = 0U, .frameTime = 10U, .cutOffFreq = kPDM_HwvadHpfBypassed, .enableFrameEnergy = true, .enablePreFilter = true, };

static const pdm_hwvad_noise_filter_t noiseFilterConfig = { .enableAutoNoiseFilter = true, .enableNoiseMin = false, .enableNoiseDecimation = false, .noiseFilterAdjustment = 0U, .noiseGain = 7U, .enableNoiseDetectOR = false, }; code param base PDM base pointer. param hwvadConfig internal filter status. param noiseConfig Voice activity detector noise filter configure structure pointer. param zcdConfig Voice activity detector zero cross detector configure structure pointer . param signalGain signal gain value, signal gain value should be properly according to application.

void PDM_EnableHwvadInterruptCallback(PDM_Type *base, *pdm_hwvad_callback_t* vadCallback, void *userData, bool enable)

Enable/Disable hwvad callback.

This function enable/disable the hwvad interrupt for the selected PDM peripheral.

Parameters

- base – Base address of the PDM peripheral.
- vadCallback – callback Pointer to store callback function, should be NULL when disable.
- userData – user data.
- enable – true is enable, false is disable.

Return values

None. –

void PDM_TransferCreateHandle(PDM_Type *base, *pdm_handle_t* *handle, *pdm_transfer_callback_t* callback, void *userData)

Initializes the PDM handle.

This function initializes the handle for the PDM transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

status_t PDM_TransferSetChannelConfig(PDM_Type *base, *pdm_handle_t* *handle, uint32_t channel, const *pdm_channel_config_t* *config, uint32_t format)

PDM set channel transfer config.

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- channel – PDM channel.
- config – channel config.
- format – data format, support data width configurations,_pdm_data_width.

Return values

kStatus_PDM_ChannelConfig_Failed – or kStatus_Success.

status_t PDM_TransferReceiveNonBlocking(PDM_Type *base, *pdm_handle_t* *handle, *pdm_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on PDM.

Note: This API returns immediately after the transfer initiates. Call the PDM_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_PDM_Busy, the transfer is finished.

Parameters

- base – PDM base pointer
- handle – Pointer to the pdm_handle_t structure which stores the transfer state.
- xfer – Pointer to the pdm_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_PDM_Busy – Previous receive still not finished.

void PDM_TransferAbortReceive(PDM_Type *base, *pdm_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – PDM base pointer
- handle – Pointer to the pdm_handle_t structure which stores the transfer state.

void PDM_TransferHandleIRQ(PDM_Type *base, *pdm_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – PDM base pointer.
- handle – Pointer to the pdm_handle_t structure.

FSL_PDM_DRIVER_VERSION

Version 2.9.3

PDM return status.

Values:

enumerator kStatus_PDM_Busy

PDM is busy.

enumerator kStatus_PDM_CLK_LOW

PDM clock frequency low

enumerator kStatus_PDM_FIFO_ERROR

PDM FIFO underrun or overflow

enumerator kStatus_PDM_QueueFull

PDM FIFO underrun or overflow

enumerator kStatus_PDM_Idle

PDM is idle

enumerator kStatus_PDM_Output_ERROR

PDM is output error

enumerator kStatus_PDM_ChannelConfig_Failed

PDM channel config failed

enumerator kStatus_PDM_HWVAD_VoiceDetected

PDM hwvad voice detected

enumerator kStatus_PDM_HWVAD_Error

PDM hwvad error

enum _pdm_interrupt_enable

The PDM interrupt enable flag.

Values:

enumerator kPDM_ErrorInterruptEnable

PDM channel error interrupt enable.

enumerator kPDM_FIFOInterruptEnable

PDM channel FIFO interrupt

enum _pdm_internal_status

The PDM status.

Values:

enumerator kPDM_StatusDfBusyFlag

Decimation filter is busy processing data

enumerator kPDM_StatusFrequencyLow

Mic app clock frequency not high enough

enumerator kPDM_StatusCh0FifoDataAvaliable
channel 0 fifo data reached watermark level

enumerator kPDM_StatusCh1FifoDataAvaliable
channel 1 fifo data reached watermark level

enumerator kPDM_StatusCh2FifoDataAvaliable
channel 2 fifo data reached watermark level

enumerator kPDM_StatusCh3FifoDataAvaliable
channel 3 fifo data reached watermark level

enum _pdm_channel_enable_mask
PDM channel enable mask.

Values:

enumerator kPDM_EnableChannel0
channgel 0 enable mask

enumerator kPDM_EnableChannel1
channgel 1 enable mask

enumerator kPDM_EnableChannel2
channgel 2 enable mask

enumerator kPDM_EnableChannel3
channgel 3 enable mask

enumerator kPDM_EnableChannelAll

enum _pdm_fifo_status
The PDM fifo status.

Values:

enumerator kPDM_FifoStatusUnderflowCh0
channel0 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh1
channel1 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh2
channel2 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh3
channel3 fifo status underflow

enumerator kPDM_FifoStatusOverflowCh0
channel0 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh1
channel1 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh2
channel2 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh3
channel3 fifo status overflow

enum _pdm_range_status
The PDM output status.

Values:

enumerator kPDM_RangeStatusUnderFlowCh0
channel0 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh1
channel1 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh2
channel2 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh3
channel3 range status underflow

enumerator kPDM_RangeStatusOverFlowCh0
channel0 range status overflow

enumerator kPDM_RangeStatusOverFlowCh1
channel1 range status overflow

enumerator kPDM_RangeStatusOverFlowCh2
channel2 range status overflow

enumerator kPDM_RangeStatusOverFlowCh3
channel3 range status overflow

enum _pdm_dc_remover
PDM DC remover configurations.

Values:

enumerator kPDM_DcRemoverCutOff20Hz
DC remover cut off 20HZ

enumerator kPDM_DcRemoverCutOff13Hz
DC remover cut off 13.3HZ

enumerator kPDM_DcRemoverCutOff40Hz
DC remover cut off 40HZ

enumerator kPDM_DcRemoverBypass
DC remover bypass

enum _pdm_df_quality_mode
PDM decimation filter quality mode.

Values:

enumerator kPDM_QualityModeMedium
quality mode memdium

enumerator kPDM_QualityModeHigh
quality mode high

enumerator kPDM_QualityModeLow
quality mode low

enumerator kPDM_QualityModeVeryLow0
quality mode very low0

enumerator kPDM_QualityModeVeryLow1
quality mode very low1

enumerator kPDM_QualityModeVeryLow2
quality mode very low2

enum _pdm_qulaity_mode_k_factor

PDM quality mode K factor.

Values:

enumerator kPDM_QualityModeHighKFactor

high quality mode K factor = 1 / 2

enumerator kPDM_QualityModeMediumKFactor

medium/very low0 quality mode K factor = 2 / 2

enumerator kPDM_QualityModeLowKFactor

low/very low1 quality mode K factor = 4 / 2

enumerator kPDM_QualityModeVeryLow2KFactor

very low2 quality mode K factor = 8 / 2

enum _pdm_df_output_gain

PDM decimation filter output gain.

Values:

enumerator kPDM_DfOutputGain0

Decimation filter output gain 0

enumerator kPDM_DfOutputGain1

Decimation filter output gain 1

enumerator kPDM_DfOutputGain2

Decimation filter output gain 2

enumerator kPDM_DfOutputGain3

Decimation filter output gain 3

enumerator kPDM_DfOutputGain4

Decimation filter output gain 4

enumerator kPDM_DfOutputGain5

Decimation filter output gain 5

enumerator kPDM_DfOutputGain6

Decimation filter output gain 6

enumerator kPDM_DfOutputGain7

Decimation filter output gain 7

enumerator kPDM_DfOutputGain8

Decimation filter output gain 8

enumerator kPDM_DfOutputGain9

Decimation filter output gain 9

enumerator kPDM_DfOutputGain10

Decimation filter output gain 10

enumerator kPDM_DfOutputGain11

Decimation filter output gain 11

enumerator kPDM_DfOutputGain12

Decimation filter output gain 12

enumerator kPDM_DfOutputGain13

Decimation filter output gain 13

enumerator kPDM_DfOutputGain14
Decimation filter output gain 14

enumerator kPDM_DfOutputGain15
Decimation filter output gain 15

enum _pdm_data_width
PDM data width.

Values:

enumerator kPDM_DataWwidth24
PDM data width 24bit

enumerator kPDM_DataWwidth32
PDM data width 32bit

enum _pdm_hwvad_interrupt_enable
PDM voice activity detector interrupt type.

Values:

enumerator kPDM_HwvadErrorInterruptEnable
PDM channel HWVAD error interrupt enable.

enumerator kPDM_HwvadInterruptEnable
PDM channel HWVAD interrupt

enum _pdm_hwvad_int_status
The PDM hwvad interrupt status flag.

Values:

enumerator kPDM_HwvadStatusInputSaturation
HWVAD saturation condition

enumerator kPDM_HwvadStatusVoiceDetectFlag
HWVAD voice detect interrupt triggered

enum _pdm_hwvad_hpf_config
High pass filter configure cut-off frequency.

Values:

enumerator kPDM_HwvadHpfBypassed
High-pass filter bypass

enumerator kPDM_HwvadHpfCutOffFreq1750Hz
High-pass filter cut off frequency 1750HZ

enumerator kPDM_HwvadHpfCutOffFreq215Hz
High-pass filter cut off frequency 215HZ

enumerator kPDM_HwvadHpfCutOffFreq102Hz
High-pass filter cut off frequency 102HZ

enum _pdm_hwvad_filter_status
HWVAD internal filter status.

Values:

enumerator kPDM_HwvadInternalFilterNormalOperation
internal filter ready for normal operation

enumerator kPDM_HwvadInternalFilterInitial
interla filter are initial

enum _pdm_hwvad_zcd_result
PDM voice activity detector zero cross detector result.

Values:

enumerator kPDM_HwvadResultOREnergyBasedDetection
zero cross detector result will be OR with energy based detection

enumerator kPDM_HwvadResultANDEnergyBasedDetection
zero cross detector result will be AND with energy based detection

typedef enum *_pdm_dc_remover* pdm_dc_remover_t
PDM DC remover configurations.

typedef enum *_pdm_df_quality_mode* pdm_df_quality_mode_t
PDM decimation filter quality mode.

typedef enum *_pdm_df_output_gain* pdm_df_output_gain_t
PDM decimation filter output gain.

typedef struct *_pdm_channel_config* pdm_channel_config_t
PDM channel configurations.

typedef struct *_pdm_config* pdm_config_t
PDM user configuration structure.

typedef enum *_pdm_hwvad_hpf_config* pdm_hwvad_hpf_config_t
High pass filter configure cut-off frequency.

typedef enum *_pdm_hwvad_filter_status* pdm_hwvad_filter_status_t
HWVAD internal filter status.

typedef struct *_pdm_hwvad_config* pdm_hwvad_config_t
PDM voice activity detector user configuration structure.

typedef struct *_pdm_hwvad_noise_filter* pdm_hwvad_noise_filter_t
PDM voice activity detector noise filter user configuration structure.

typedef enum *_pdm_hwvad_zcd_result* pdm_hwvad_zcd_result_t
PDM voice activity detector zero cross detector result.

typedef struct *_pdm_hwvad_zero_cross_detector* pdm_hwvad_zero_cross_detector_t
PDM voice activity detector zero cross detector configuration structure.

typedef struct *_pdm_transfer* pdm_transfer_t
PDM SDMA transfer structure.

typedef struct *_pdm_handle* pdm_handle_t
PDM handle.

typedef void (*pdm_transfer_callback_t)(PDM_Type *base, *pdm_handle_t* *handle, *status_t* status, void *userData)
PDM transfer callback prototype.

typedef void (*pdm_hwvad_callback_t)(*status_t* status, void *userData)
PDM HWVAD callback prototype.

typedef struct *_pdm_hwvad_notification* pdm_hwvad_notification_t
PDM HWVAD notification structure.

PDM_XFER_QUEUE_SIZE
PDM XFER QUEUE SIZE.

struct _pdm_channel_config
#include <fsl_pdm.h> PDM channel configurations.

Public Members

pdm_dc_remover_t outputCutOffFreq
PDM output DC remover cut off frequency

pdm_df_output_gain_t gain
Decimation Filter Output Gain

struct _pdm_config
#include <fsl_pdm.h> PDM user configuration structure.

Public Members

bool enableDoze
This module will enter disable/low leakage mode if DOZEN is active with ipg_doze is asserted

bool enableFilterBypass
Switchable bypass path for the decimation filter

uint8_t fifoWatermark
Watermark value for FIFO

pdm_df_quality_mode_t qualityMode
Quality mode

uint8_t cicOverSampleRate
CIC filter over sampling rate

struct _pdm_hwvad_config
#include <fsl_pdm.h> PDM voice activity detector user configuration structure.

Public Members

uint8_t channel
Which channel uses voice activity detector

uint8_t initializeTime
Number of frames or samples to initialize voice activity detector.

uint8_t cicOverSampleRate
CIC filter over sampling rate

uint8_t inputGain
Voice activity detector input gain

uint32_t frameTime
Voice activity frame time

pdm_hwvad_hpf_config_t cutOffFreq
High pass filter cut off frequency

bool enableFrameEnergy
If frame energy enabled, true means enable

bool enablePreFilter
If pre-filter enabled

struct __pdm_hwvad_noise_filter
#include <fsl_pdm.h> PDM voice activity detector noise filter user configuration structure.

Public Members

bool enableAutoNoiseFilter
If noise fileter automatically activated, true means enable

bool enableNoiseMin
If Noise minimum block enabled, true means enabled

bool enableNoiseDecimation
If enable noise input decimation

bool enableNoiseDetectOR
Enables a OR logic in the output of minimum noise estimator block

uint32_t noiseFilterAdjustment
The adjustment value of the noise filter

uint32_t noiseGain
Gain value for the noise energy or envelope estimated

struct __pdm_hwvad_zero_cross_detector
#include <fsl_pdm.h> PDM voice activity detector zero cross detector configuration structure.

Public Members

bool enableAutoThreshold
If ZCD auto-threshold enabled, true means enabled.

pdm_hwvad_zcd_result_t zcdAnd
Is ZCD result is AND'ed with energy-based detection, false means OR'ed

uint32_t threshold
The adjustment value of the noise filter

uint32_t adjustmentThreshold
Gain value for the noise energy or envelope estimated

struct __pdm_transfer
#include <fsl_pdm.h> PDM SDMA transfer structure.

Public Members

volatile uint8_t *data
Data start address to transfer.

volatile size_t dataSize
Total Transfer bytes size.

struct __pdm__hwvad__notification

#include <fsl_pdm.h> PDM HWVAD notification structure.

struct __pdm__handle

#include <fsl_pdm.h> PDM handle structure.

Public Members

uint32_t state

Transfer status

pdm_transfer_callback_t callback

Callback function called at transfer event

void *userData

Callback parameter passed to callback function

pdm_transfer_t pdmQueue[(4U)]

Transfer queue storing queued transfer

size_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

uint32_t format

data format

uint8_t watermark

Watermark value

uint8_t startChannel

end channel

uint8_t channelNums

Enabled channel number

2.66 PDM EDMA Driver

void PDM__TransferInstallEDMATCDMemory(*pdm_edma_handle_t* *handle, void *tcdAddr, size_t tcdNum)

Install EDMA descriptor memory.

Parameters

- handle – Pointer to EDMA channel transfer handle.
- tcdAddr – EDMA head descriptor address.
- tcdNum – EDMA link descriptor address.

void PDM__TransferCreateHandleEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, *pdm_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the PDM Rx eDMA handle.

This function initializes the PDM slave DMA handle, which can be used for other PDM master transactional APIs. Usually, for a specified PDM instance, call this API once to get the initialized handle.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void PDM_TransferSetMultiChannelInterleaveType(*pdm_edma_handle_t* *handle, *pdm_edma_multi_channel_interleave_t* multiChannelInterleaveType)

Initializes the multi PDM channel interleave type.

This function initializes the PDM DMA handle member interleaveType, it shall be called only when application would like to use type kPDM_EDMAMultiChannelInterleavePerChannelBlock, since the default interleaveType is kPDM_EDMAMultiChannelInterleavePerChannelSample always

Parameters

- handle – PDM eDMA handle pointer.
- multiChannelInterleaveType – Multi channel interleave type.

void PDM_TransferSetChannelConfigEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, uint32_t channel, const *pdm_channel_config_t* *config)

Configures the PDM channel.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- channel – channel index.
- config – pdm channel configurations.

status_t PDM_TransferReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, *pdm_edma_transfer_t* *xfer)

Performs a non-blocking PDM receive using eDMA.

Mcaro MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL can control whether PDM is enabled internally or externally.

a. Scatter gather case: This functio support dynamic scatter gather and staic scatter gather, a. for the dynamic scatter gather case: Application should call PDM_TransferReceiveEDMA function continuously to make sure new receive request is submit before the previous one finish. b. for the static scatter gather case: Application should use the link transfer feature and make sure a loop link transfer is provided, such as:

```
pdm_edma_transfer_t pdmXfer[2] =
{
    {
    .data  = s_buffer,
    .dataSize = BUFFER_SIZE,
    .linkTransfer = &pdmXfer[1],
    },

    {
    .data  = &s_buffer[BUFFER_SIZE],
    .dataSize = BUFFER_SIZE,
    .linkTransfer = &pdmXfer[0]
    },
};
```

b. Multi channel case: This function support receive multi pdm channel data, for example, if two channel is requested,

```
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_0, &channelConfig);
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_1, &channelConfig);
PDM_TransferReceiveEDMA(DEMO_PDM, &s_pdmRxHandle_0, pdmXfer);
```

The output data will be formatted as below if handle->interleaveType =

Note: This interface returns immediately after the transfer initiates. Call the PDM_GetReceiveRemainingBytes to poll the transfer status and check whether the PDM transfer is finished.

---

void PDM_TransferTerminateReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Terminate all PDM receive.

This function will clear all transfer slots buffered in the pdm queue. If users only want to abort the current transfer slot, please call PDM_TransferAbortReceiveEDMA.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.

void PDM_TransferAbortReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Aborts a PDM receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call PDM_TransferTerminateReceiveEDMA.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

status_t PDM_TransferGetReceiveCountEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, size_t *count)

Gets byte count received by PDM.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

- count – Bytes count received by PDM.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

FSL_PDM_EDMA_DRIVER_VERSION

Version 2.6.5

enum _pdm_edma_multi_channel_interleave

pdm multi channel interleave type

Values:

enumerator kPDM_EDMAMultiChannelInterleavePerChannelSample

enumerator kPDM_EDMAMultiChannelInterleavePerChannelBlock

typedef struct *_pdm_edma_handle* pdm_edma_handle_t

PDM edma handler.

typedef enum *_pdm_edma_multi_channel_interleave* pdm_edma_multi_channel_interleave_t

pdm multi channel interleave type

typedef struct *_pdm_edma_transfer* pdm_edma_transfer_t

PDM edma transfer.

typedef void (*pdm_edma_callback_t)(PDM_Type *base, *pdm_edma_handle_t* *handle, *status_t* status, void *userData)

PDM eDMA transfer callback function for finish and error.

MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL

the PDM enable position When calling PDM_TransferReceiveEDMA

struct _pdm_edma_transfer

#include <fsl_pdm_edma.h> PDM edma transfer.

Public Members

volatile uint8_t *data

Data start address to transfer.

volatile size_t dataSize

Total Transfer bytes size.

struct *_pdm_edma_transfer* *linkTransfer

linked transfer configurations

struct _pdm_edma_handle

#include <fsl_pdm_edma.h> PDM DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for PDM send

uint8_t count

The transfer data count in a DMA request

uint32_t receivedBytes
: total transfer count

uint32_t state
: Internal state for PDM eDMA transfer

pdm_edma_callback_t callback
: Callback for users while transfer finish or error occurs

bool isLoopTransfer
: loop transfer

void *userData
: User callback parameter

edma_tcd_t *tcd
: TCD pool for eDMA transfer.

uint32_t tcdNum
: TCD number

uint32_t tcdUser
: Index for user to queue transfer.

uint32_t tcdDriver
: Index for driver to get the transfer data and size

volatile uint32_t tcdUsedNum
: Index for user to queue transfer.

pdm_edma_multi_channel_interleave_t interleaveType
: multi channel transfer interleave type

uint8_t endChannel
: The last enabled channel

uint8_t channelNums
: total channel numbers

2.67 PINT: Pin Interrupt and Pattern Match Driver

FSL_PINT_DRIVER_VERSION

enum _pint_pin_enable
: PINT Pin Interrupt enable type.

Values:

enumerator kPINT_PinIntEnableNone
: Do not generate Pin Interrupt

enumerator kPINT_PinIntEnableRiseEdge
: Generate Pin Interrupt on rising edge

enumerator kPINT_PinIntEnableFallEdge
: Generate Pin Interrupt on falling edge

enumerator kPINT_PinIntEnableBothEdges
: Generate Pin Interrupt on both edges

enumerator kPINT_PinIntEnableLowLevel
Generate Pin Interrupt on low level

enumerator kPINT_PinIntEnableHighLevel
Generate Pin Interrupt on high level

enum _pint_int
PINT Pin Interrupt type.

Values:

enumerator kPINT_PinInt0
Pin Interrupt 0

enum _pint_pmatch_input_src
PINT Pattern Match bit slice input source type.

Values:

enumerator kPINT_PatternMatchInp0Src
Input source 0

enumerator kPINT_PatternMatchInp1Src
Input source 1

enumerator kPINT_PatternMatchInp2Src
Input source 2

enumerator kPINT_PatternMatchInp3Src
Input source 3

enumerator kPINT_PatternMatchInp4Src
Input source 4

enumerator kPINT_PatternMatchInp5Src
Input source 5

enumerator kPINT_PatternMatchInp6Src
Input source 6

enumerator kPINT_PatternMatchInp7Src
Input source 7

enumerator kPINT_SecPatternMatchInp0Src
Input source 0

enumerator kPINT_SecPatternMatchInp1Src
Input source 1

enum _pint_pmatch_bslice
PINT Pattern Match bit slice type.

Values:

enumerator kPINT_PatternMatchBSlice0
Bit slice 0

enum _pint_pmatch_bslice_cfg
PINT Pattern Match configuration type.

Values:

enumerator kPINT_PatternMatchAlways
Always Contributes to product term match

enumerator kPINT_PatternMatchStickyRise
Sticky Rising edge

enumerator kPINT_PatternMatchStickyFall
Sticky Falling edge

enumerator kPINT_PatternMatchStickyBothEdges
Sticky Rising or Falling edge

enumerator kPINT_PatternMatchHigh
High level

enumerator kPINT_PatternMatchLow
Low level

enumerator kPINT_PatternMatchNever
Never contributes to product term match

enumerator kPINT_PatternMatchBothEdges
Either rising or falling edge

typedef enum *_pint_pin_enable* pint_pin_enable_t
PINT Pin Interrupt enable type.

typedef enum *_pint_int* pint_pin_int_t
PINT Pin Interrupt type.

typedef enum *_pint_pmatch_input_src* pint_pmatch_input_src_t
PINT Pattern Match bit slice input source type.

typedef enum *_pint_pmatch_bslice* pint_pmatch_bslice_t
PINT Pattern Match bit slice type.

typedef enum *_pint_pmatch_bslice_cfg* pint_pmatch_bslice_cfg_t
PINT Pattern Match configuration type.

typedef struct *_pint_status* pint_status_t
PINT event status.

typedef void (*pint_cb_t)(*pint_pin_int_t* pintr, *pint_status_t* *status)
PINT Callback function.

typedef struct *_pint_pmatch_cfg* pint_pmatch_cfg_t

void PINT_Init(PINT_Type *base)
Initialize PINT peripheral.

This function initializes the PINT peripheral and enables the clock.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_SetCallback(PINT_Type *base, *pint_cb_t* callback)
Set PINT callback.

This function set the callback for PINT interupt handler.

Parameters

- base – Base address of the PINT peripheral.
- callback – Callback.

Return values
None. –

void PINT_PinInterruptConfig(PINT_Type *base, *pint_pin_int_t* intr, *pint_pin_enable_t* enable)

Configure PINT peripheral pin interrupt.

This function configures a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- intr – Pin interrupt.
- enable – Selects detection logic.

Return values
None. –

void PINT_PinInterruptGetConfig(PINT_Type *base, *pint_pin_int_t* pintr, *pint_pin_enable_t* *enable)

Get PINT peripheral pin interrupt configuration.

This function returns the configuration of a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.
- enable – Pointer to store the detection logic.

Return values
None. –

void PINT_PinInterruptClrStatus(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.

This function clears the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetStatus(PINT_Type *base, *pint_pin_int_t* pintr)

Get Selected pin interrupt status.

This function returns the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
status – = 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

void PINT_PinInterruptClrStatusAll(PINT_Type *base)

Clear all pin interrupts status only when pins were triggered by edge-sensitive.

This function clears the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetStatusAll(PINT_Type *base)

Get all pin interrupts status.

This function returns the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
status – Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active.

static inline void PINT_PinInterruptClrFallFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt fall flag.

This function clears the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetFallFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Get selected pin interrupt fall flag.

This function returns the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
flag – = 0 Falling edge has not been detected. = 1 Falling edge has been detected.

static inline void PINT_PinInterruptClrFallFlagAll(PINT_Type *base)

Clear all pin interrupt fall flags.

This function clears the fall flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetFallFlagAll(PINT_Type *base)

Get all pin interrupt fall flags.

This function returns the fall flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
flags – Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

static inline void PINT_PinInterruptClrRiseFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt rise flag.

This function clears the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

None. –

static inline uint32_t PINT_PinInterruptGetRiseFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Get selected pin interrupt rise flag.

This function returns the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

flag – = 0 Rising edge has not been detected. = 1 Rising edge has been detected.

static inline void PINT_PinInterruptClrRiseFlagAll(PINT_Type *base)

Clear all pin interrupt rise flags.

This function clears the rise flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline uint32_t PINT_PinInterruptGetRiseFlagAll(PINT_Type *base)

Get all pin interrupt rise flags.

This function returns the rise flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

flags – Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

void PINT_PatternMatchConfig(PINT_Type *base, *pint_pmatch_bslice_t* bslice, *pint_pmatch_cfg_t* *cfg)

Configure PINT pattern match.

This function configures a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

void PINT_PatternMatchGetConfig(PINT_Type *base, *pint_pmatch_bslice_t* bslice, *pint_pmatch_cfg_t* *cfg)

Get PINT pattern match configuration.

This function returns the configuration of a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

static inline uint32_t PINT_PatternMatchGetStatus(PINT_Type *base, *pint_pmatch_bslice_t* bslice)

Get pattern match bit slice status.

This function returns the status of selected bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.

Return values

status – = 0 Match has not been detected. = 1 Match has been detected.

static inline uint32_t PINT_PatternMatchGetStatusAll(PINT_Type *base)

Get status of all pattern match bit slices.

This function returns the status of all bit slices.

Parameters

- base – Base address of the PINT peripheral.

Return values

status – Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected.

uint32_t PINT_PatternMatchResetDetectLogic(PINT_Type *base)

Reset pattern match detection logic.

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

- base – Base address of the PINT peripheral.

Return values

pmstatus – Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.

static inline void PINT_PatternMatchEnable(PINT_Type *base)

Enable pattern match function.

This function enables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchDisable(PINT_Type *base)

Disable pattern match function.

This function disables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchEnableRXEV(PINT_Type *base)

Enable RXEV output.

This function enables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchDisableRXEV(PINT_Type *base)

Disable RXEV output.

This function disables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

void PINT_EnableCallback(PINT_Type *base)

Enable callback.

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

void PINT_DisableCallback(PINT_Type *base)

Disable callback.

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

- base – Base address of the peripheral.

Return values

None. –

void PINT_Deinit(PINT_Type *base)

Deinitialize PINT peripheral.

This function disables the PINT clock.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_EnableCallbackByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)
enable callback by pin index.

This function enables callback by pin index instead of enabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values
None. –

void PINT_DisableCallbackByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)
disable callback by pin index.

This function disables callback by pin index instead of disabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values
None. –

PINT_USE_LEGACY_CALLBACK

PININT_BITSLICE_SRC_START

PININT_BITSLICE_SRC_MASK

PININT_BITSLICE_CFG_START

PININT_BITSLICE_CFG_MASK

PININT_BITSLICE_ENDP_MASK

PINT_PIN_INT_LEVEL

PINT_PIN_INT_EDGE

PINT_PIN_INT_FALL_OR_HIGH_LEVEL

PINT_PIN_INT_RISE

PINT_PIN_RISE_EDGE

PINT_PIN_FALL_EDGE

PINT_PIN_BOTH_EDGE

PINT_PIN_LOW_LEVEL

PINT_PIN_HIGH_LEVEL

struct _pint_status
#include <fsl_pint.h> PINT event status.

struct _pint_pmatch_cfg
#include <fsl_pint.h>

2.68 PNGDEC: PNG decoder Driver

Error codes for the PNGDEC driver.

Values:

enumerator kStatus_PNGDEC_NotSupported
Interlace data chunk is not supported.

enumerator kStatus_PNGDEC_WidthTooLarge
The image width is larger than 1024.

enum _pngdec_interrupt_enable
Interrupt enable/disable mask.

Values:

enumerator kPNGDEC_ChecksumErrorFlag
ADLER-32 checksum error status and interrupt enable.

enumerator kPNGDEC_CrcErrorFlag
CRC error status and interrupt enable.

enumerator kPNGDEC_HeaderErrorFlag
Header error status and interrupt enable.

enumerator kPNGDEC_BtypeErrorFlag
B type error status and interrupt enable. TODO

enumerator kPNGDEC_ZlibHeaderErrorFlag
Zlib header error status and interrupt enable.

enumerator kPNGDEC_BitDepthErrorFlag
Bit depth error status and interrupt enable. Bit depth less than 8 bit not supported.

enumerator kPNGDEC_InterlaceErrorFlag
Interlace error status and interrupt enable. Interlace is not supported.

enumerator kPNGDEC_WidthErrorFlag
Width error status and interrupt enable. Max image width 1024.

enumerator kPNGDEC_EncodeDataDoneFlag
Data encoding done status and interrupt enable.

enumerator kPNGDEC_DecodePixelDoneFlag
Pixel data decoding done status and interrupt enable.

enumerator kPNGDEC_DecodeAncillaryDoneFlag
Ancillary data decoding done status and interrupt enable.

enumerator kPNGDEC_ErrorFlags

enum _pngdec_endian_mode
PNG decode endian mode.

Values:

enumerator kPNGDEC_LittleEndian
Little endian.

enumerator kPNGDEC_BigEndian
Big endian.

enum _pngdec_color_type

PNG color type.

Values:

enumerator kPNGDEC_PixelFormatY8

8-bit monochrome(gray scale).

enumerator kPNGDEC_PixelFormatYA88

16-bit monochrome(gray scale) with alpha.

enumerator kPNGDEC_PixelFormatRGB888

24-bit RGB true color, 8-bit each palette.

enumerator kPNGDEC_PixelFormatRGBA8888

32-bit RGBA true color with alpha, 8-bit each palette.

enumerator kPNGDEC_PixelFormatRGB16_16_16

48-bit RGB true color, 16-bit each palette.

enumerator kPNGDEC_PixelFormatRGBA16_16_16_16

64-bit RGBA true color with alpha, 16-bit each palette.

enumerator kPNGDEC_PixelFormatLUT8

8-bit Indexed Color.

typedef enum *_pngdec_endian_mode* pngdec_endian_mode_t

PNG decode endian mode.

typedef struct *_pngdec_config* pngdec_config_t

Configuration for PNGDEC.

typedef enum *_pngdec_color_type* pngdec_color_type_t

PNG color type.

typedef struct *_pngdec_image* pngdec_image_t

The decoded image info structure.

FSL_PNGDEC_DRIVER_VERSION

PNGDEC driver version.

void PNGDEC_Init(PNGDEC_Type *base, const *pngdec_config_t* *config)

Initializes the PNGDEC.

The default configuration can be got by calling PNGDEC_GetDefaultConfig().

Parameters

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

void PNGDEC_Deinit(PNGDEC_Type *base)

Deinitializes the PNGDEC.

Parameters

- base – PNGDEC peripheral base address.

void PNGDEC_GetDefaultConfig(*pngdec_config_t* *config)

Gets the default configuration for PNGDEC.

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

Example:

```
config->enableAncillary = false;
config->enable = false;
```

Parameters

- config – Pointer to PNGDEC configuration structure.

static inline void PNGDEC_EnableInterrupts(PNGDEC_Type *base, uint32_t mask)

Enables interrupts.

Parameters

- base – PNGDEC peripheral base address.
- mask – Interrupts mask. See "_pngdec_interrupt_enable".

static inline void PNGDEC_DisableInterrupts(PNGDEC_Type *base, uint32_t mask)

Disables interrupts.

Parameters

- base – PNGDEC peripheral base address.
- mask – Interrupts mask. See "_pngdec_interrupt_enable".

static inline uint32_t PNGDEC_GetStatusFlags(PNGDEC_Type *base)

Gets status flags.

Parameters

- base – PNGDEC peripheral base address.

Returns

Status flags asserted mask. See "_pngdec_status_flags".

static inline void PNGDEC_ClearStatusFlags(PNGDEC_Type *base, uint32_t mask)

Clears status flags.

Parameters

- base – PNGDEC peripheral base address.
- mask – Status flags mask. See "_pngdec_status_flags".

static inline void PNGDEC_Reset(PNGDEC_Type *base)

Resets the PNGDEC code and DMA logic.

Parameters

- base – PNGDEC peripheral base address.

static inline void PNGDEC_Enable(PNGDEC_Type *base, bool enable)

Enables or disables the PNGDEC.

Parameters

- base – PNGDEC peripheral base address.
- enable – True to enable the PNGDEC.

static inline void PNGDEC_EnableAncillary(PNGDEC_Type *base, bool enable)

Enables or disables the PNGDEC ancillary drop.

Parameters

- base – PNGDEC peripheral base address.
- enable – True to enable the PNGDEC ancillary drop.

static inline void PNGDEC_StartDecode(PNGDEC_Type *base)

Starts the PNG decode.

Parameters

- base – PNGDEC peripheral base address.

void PNGDEC_SetPngBuffer(PNGDEC_Type *base, uint8_t *buffer, size_t length)

Sets the address and length of the raw PNG image.

Parameters

- base – PNGDEC peripheral base address.
- data – Start address of the buffer of the raw PNG image, shall be 8-byte aligned.
- length – Size of the buffer.

static inline void PNGDEC_SetOutputBuffer(PNGDEC_Type *base, uint8_t *imageData, uint8_t *ancillaryData)

Sets the address of the decoded data.

Parameters

- base – PNGDEC peripheral base address.
- imageData – Start address of the buffer for the decoded image data, shall be 8-byte aligned.
- ancillaryData – Start address of the buffer for the decoded ancillary data, shall be 8-byte aligned.

status_t PNGDEC_ParseHeader(*pngdec_image_t* *image, uint8_t *pngBuf)

Parses the PNG header and stores the info in the decoded image info structure.

Parameters

- image – Pointer to PNGDEC decoded image info structure.
- pngBuf – Pointer to PNG file buffer.

Return values

- kStatus_Success – Header parsing success.
- kStatus_Fail – PNG header parsing failed due corrupted header.
- kStatus_PNGDEC_WidthTooLarge – Header parsing failed due to the image width is larger than 1024.
- kStatus_PNGDEC_NotSupported – Header parsing failed due to the image is interlaced or the bit depth is less than 8.

status_t PNGDEC_Decode(PNGDEC_Type *base, *pngdec_image_t* *image, uint32_t *status)

Decodes the PNG image.

This function performs the PNG decoding in blocking way and stores the decoded info in decoded image info structure.

Parameters

- base – PNGDEC peripheral base address.
- image – Pointer to PNGDEC decoded image info structure.
- status – Pointer to decoded status. When retval is kStatus_Fail, Checksum/Crc/Header/Btype/ZlibHeader error may occur due to PNG file corruption, user can check which error(s) occured if necessary.

Return values

- kStatus_Success – PNG decoding success.
- kStatus_Fail – PNG decoding failed due to CRC/header/B type/Alder error or invalid PNG file.
- kStatus_PNGDEC_WidthTooLarge – PNG decoding failed due to the image width is larger than 1024.
- kStatus_PNGDEC_NotSupported – PNG decoding failed due to the image is interlaced or the bit depth is less than 8.

bool enableAncillary
: Enable ancillary data drop.

bool enable
: Enables the decode IP.

uint32_t width
: Decoded image width.

uint32_t height
: Decoded image height.

pngdec_color_type_t colorType
: Color type for the generated image.

uint8_t *imageData
: Pointer to the buffer for decoded image data.

uint8_t *ancillaryData
: Pointer to the buffer for decoded ancillary data.

uint32_t ancillaryLength
: Length of the ancillary data.

struct __pngdec__config
: *#include <fsl_pngdec.h>* Configuration for PNGDEC.

struct __pngdec__image
: *#include <fsl_pngdec.h>* The decoded image info structure.

2.69 Power Driver

enum pd__bits
: Power down control bits defined in SLEEPCON and PMC.

  Values:

  enumerator kPDRUNCFG__SHUT__SENSEP__MAINCLK
  : The Private Part in VDD1_SENSE Main Clock Shut Off.

  enumerator kPDRUNCFG__SHUT__SENSES__MAINCLK
  : The Shared Part in VDD1_SENSE Main Clock Shut Off.

  enumerator kPDRUNCFG__SHUT__RAM0__CLK
  : Main Clock of RAM Arbiter 0 Shut Off.

  enumerator kPDRUNCFG__SHUT__RAM1__CLK
  : Main Clock of RAM Arbiter 1 shut off.

enumerator kPDRUNCFG_SHUT_COMNN_MAINCLK
VDDN_COM Domain Main Clock Shut Off.

enumerator kPDRUNCFG_SHUT_MEDIA_MAINCLK
VDD2_MEDIA and VDDN_MEDIA Domains Main Clock of Shut Off.

enumerator kPDRUNCFG_PD_SYSXTAL
Xtal Oscillator Power Down.

enumerator kPDRUNCFG_PD_FRO2
FRO2 Power Down.

enumerator kPDRUNCFG_PD_LPOSC
LPOSC Power Down.

enumerator kPDRUNCFG_PD_PLLANA
Main PLL Analog Function Power Down.

enumerator kPDRUNCFG_PD_PLLLDO
Main PLL Internal Regulator Power Down.

enumerator kPDRUNCFG_PD_AUDPLLANA
Audio PLL Analog Function Power Down.

enumerator kPDRUNCFG_PD_AUDPLLLDO
Audio PLL Internal Regulator Power Down.

enumerator kPDRUNCFG_PD_ADC0
ADC0 Power Down, 1 - Applies gating.

enumerator kPDRUNCFG_GATE_FRO2
FRO2 Gating, 1 - Applies gating.

enumerator kPDRUNCFG_DSR_VDD2N_MEDIA
Power Switch and DSR Enable for the VDD2_MEDIA and VDDN_MEDIA domains.

enumerator kPDRUNCFG_DSR_VDDN_COM
Power Switch and DSR Enable for the VDDN_COM domain.

enumerator kPDRUNCFG_PD_VDD2_DSP
Power switch for the HiFi4 DSP.

enumerator kPDRUNCFG_PD_VDD2_MIPI
Power Switch for the MIPI PHY.

enumerator kPDRUNCFG_PD_RBB_VDD1
Power Down RBB in VDD1.

enumerator kPDRUNCFG_PD_AFBB_VDD1
Power Down AFBB in VDD1 Domain.

enumerator kPDRUNCFG_PD_RBB_VDD2
Power Down RBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_AFBB_VDD2
Power Down AFBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_RBB_VDDN
Power Down RBB in VDDN Domain.

enumerator kPDRUNCFG_PD_AFBB_VDDN
Power Down RBB in VDDN Domain.

enumerator kPDRUNCFG_PD_SRAM_RBB_VDD1
Power Down SRAM RBB in VDD1 Domain.

enumerator kPDRUNCFG_PD_SRAM_RBB_VDD2
Power Down SRAM RBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_SRAM_AFBB_VDD2
Power Down SRAM AFBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_PMC_TEMPSNS
Power Down PMC Temperature Sensor.

enumerator kPDRUNCFG_LP_PMCREF
PMC References in Low-Power Mode.

enumerator kPDRUNCFG_PD_HVD1V8
HVD Power Down.

enumerator kPDRUNCFG_LP_POR_VDD1
POR Low-Power Mode in VDD1 Domain.

enumerator kPDRUNCFG_LP_LVD_VDD1
LVD Low-Power Mode in VDD1 Domain.

enumerator kPDRUNCFG_PD_HVD_VDD1
Power Down HVD in VDD1 Domain.

enumerator kPDRUNCFG_PD_AGDET_VDD1
Power Down AGDET in VDD1 Domain.

enumerator kPDRUNCFG_LP_POR_VDD2
POR Low-Power Mode in VDD2 Domain.

enumerator kPDRUNCFG_LP_LVD_VDD2
LVD Low-Power Mode in VDD2 Domain.

enumerator kPDRUNCFG_PD_HVD_VDD2
Power Down HVD in VDD2 Domain.

enumerator kPDRUNCFG_PD_AGDET_VDD2
Power Down AGDET in VDD2 Domain.

enumerator kPDRUNCFG_LP_POR_VDDN
POR Low-Power Mode in VDDN Domain.

enumerator kPDRUNCFG_LP_LVD_VDDN
LVD Low-Power Mode in VDDN Domain.

enumerator kPDRUNCFG_PD_HVD_VDDN
Power Down HVD in VDDN Domain.

enumerator kPDRUNCFG_PD_OTP
Power Down OTP.

enumerator kPDRUNCFG_PD_ROM
Power Down ROM.

enumerator kPDRUNCFG_APD_SRAM1
RAM Partition 1 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM2
RAM Partition 2 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM3
RAM Partition 3 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM4
RAM Partition 4 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM5
RAM Partition 5 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM6
RAM Partition 6 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM7
RAM Partition 7 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM8
RAM Partition 8 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM9
RAM Partition 9 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM10
RAM Partition 10 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM11
RAM Partition 11 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM12
RAM Partition 12 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM13
RAM Partition 13 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM14
RAM Partition 14 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM15
RAM Partition 15 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM16
RAM Partition 16 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM17
RAM Partition 17 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM18
RAM Partition 18 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM19
RAM Partition 19 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM20
RAM Partition 20 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM21
RAM Partition 21 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM22
RAM Partition 22 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM23
RAM Partition 23 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM24
RAM Partition 24 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM25
RAM Partition 25 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM26
RAM Partition 26 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM27
RAM Partition 27 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM28
RAM Partition 28 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM29
RAM Partition 29 Array Power Down.

enumerator kPDRUNCFG_PPD_SRAM1
RAM Partition 1 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM2
RAM Partition 2 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM3
RAM Partition 3 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM4
RAM Partition 4 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM5
RAM Partition 5 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM6
RAM Partition 6 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM7
RAM Partition 7 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM8
RAM Partition 8 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM9
RAM Partition 9 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM10
RAM Partition 10 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM11
RAM Partition 11 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM12
RAM Partition 12 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM13
RAM Partition 13 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM14
RAM Partition 14 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM15
RAM Partition 15 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM16
RAM Partition 16 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM17
RAM Partition 17 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM18
RAM Partition 18 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM19
RAM Partition 19 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM20
RAM Partition 20 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM21
RAM Partition 21 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM22
RAM Partition 22 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM23
RAM Partition 23 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM24
RAM Partition 24 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM25
RAM Partition 25 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM26
RAM Partition 26 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM27
RAM Partition 27 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM28
RAM Partition 28 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM29
RAM Partition 29 Periphery Power Down.

enumerator kPDRUNCFG_APD_SDHC0_SRAM
uSDHC0 SRAM array Power Down.

enumerator kPDRUNCFG_APD_SDHC1_SRAM
uSDHC1 SRAM array Power Down.

enumerator kPDRUNCFG_APD_USB0_SRAM
USB0 SRAM array Power Down.

enumerator kPDRUNCFG_APD_USB1_SRAM
USB1 SRAM array Power Down.

enumerator kPDRUNCFG_APD_JPEGDEC
Powers down JPEGDEC RAMs array.

enumerator kPDRUNCFG_APD_PNGDEC
Powers down PNGDEC RAMs array.

enumerator kPDRUNCFG_APD_MIPIDSI
Powers down MIPI PHY RAMs array.

enumerator kPDRUNCFG_APD_GPU
Powers down VGPU RAMs array.

enumerator kPDRUNCFG_APD_DMA2_3
Powers down DMA2 and DMA3 RAMs array.

enumerator kPDRUNCFG_APD_DMA0_1_PKC_ETF
Powers down DMA0-1, PKC, and ETF RAMs array.

enumerator kPDRUNCFG_APD_CPU0_CODE_CACHE
Powers down CPU0 code cache RAMs array.

enumerator kPDRUNCFG_APD_CPU0_SYSTEM_CACHE
Powers down CPU0 system cache RAMs array.

enumerator kPDRUNCFG_APD_DSP_ICACHE
Powers down HiFi4 instruction cache RAMs array.

enumerator kPDRUNCFG_APD_DSP_DCACHE
Powers down HiFi4 data cache RAMs array.

enumerator kPDRUNCFG_APD_DSP_ITCM
Powers down HiFi4 instruction TCM RAMs array.

enumerator kPDRUNCFG_APD_DSP_DTCM
Powers down HiFi4 data TCM RAMs array.

enumerator kPDRUNCFG_APD_EZHV_TCM
Powers down EZH-V TCM RAMs array.

enumerator kPDRUNCFG_APD_NPU
Powers down NPU RAMs array.

enumerator kPDRUNCFG_APD_XSPI0
Powers down XSPI0, MMU0, and cache RAMs array.

enumerator kPDRUNCFG_APD_XSPI1
Powers down XSPI1, MMU1, and cache RAMs array.

enumerator kPDRUNCFG_APD_XSPI2
Powers down XSPI2 and MMU2 RAMs array.

enumerator kPDRUNCFG_APD_LCDIF
Powers down LCDIF controller RAMs array.

enumerator kPDRUNCFG_APD_OCOTP
Powers down OCOTP shadow RAMs array.

enumerator kPDRUNCFG_PPD_SDHC0_SRAM
Powers down uSDHC0 RAMs periphery.

enumerator kPDRUNCFG_PPD_SDHC1_SRAM
Powers down uSDHC1 RAMs periphery.

enumerator kPDRUNCFG_PPD_USB0_SRAM
Powers down USB0 RAMs periphery.

enumerator kPDRUNCFG_PPD_USB1_SRAM
Powers down USB1 RAMs periphery.

enumerator kPDRUNCFG_PPD_JPEGDEC
Powers down JPEGDEC RAMs periphery.

enumerator kPDRUNCFG_PPD_PNGDEC
Powers down PNGDEC RAMs periphery.

enumerator kPDRUNCFG_PPD_MIPIDSI
Powers down MIPI PHY RAMs periphery.

enumerator kPDRUNCFG_PPD_GPU
Powers down VGPU RAMs periphery.

enumerator kPDRUNCFG_PPD_DMA2_3
Powers down DMA2 and DMA3 RAMs periphery.

enumerator kPDRUNCFG_PPD_DMA0_1_PKC_ETF
Powers down DMA0-1, PKC, and ETF RAMs periphery.

enumerator kPDRUNCFG_PPD_CPU0_CODE_CACHE
Powers down CPU0 code cache RAMs periphery.

enumerator kPDRUNCFG_PPD_CPU0_SYSTEM_CACHE
Powers down CPU0 system cache RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_ICACHE
Powers down HiFi4 instruction cache RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_DCACHE
Powers down HiFi4 data cache RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_ITCM
Powers down HiFi4 instruction TCM RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_DTCM
Powers down HiFi4 data TCM RAMs periphery.

enumerator kPDRUNCFG_PPD_EZHV_TCM
Powers down EZH-V TCM RAMs periphery.

enumerator kPDRUNCFG_PPD_NPU
Powers down NPU RAMs periphery.

enumerator kPDRUNCFG_PPD_XSPI0
Powers down XSPI0, MMU0, and cache RAMs periphery.

enumerator kPDRUNCFG_PPD_XSPI1
Powers down XSPI1, MMU1, and cache RAMs periphery.

enumerator kPDRUNCFG_PPD_XSPI2
Powers down XSPI2 and MMU2 RAMs periphery.

enumerator kPDRUNCFG_PPD_LCDIF
Powers down LCDIF controller RAMs periphery.

enumerator kPDRUNCFG_PPD_OCOTP
Powers down OCOTP shadow RAMs periphery.

enumerator kPDRUNCFG_ForceUnsigned

enum lp_bits
low power control bits for modules. Bits used for modules to requesting low-power.

Values:

enumerator kPower_VGPU_LPREQ
VGPU low power request.

enumerator kPower_EZHV_STOP
EZHV stop request.

enumerator kPower_EZHV_HALT
EZHV halt request.

enumerator kPower_EZHV_EXIT_WAIT
EZHV exit wait request.

enumerator kPower_XSPI2_STOP
XSPI2 stop request.

enumerator kPower_LPSPI14_B_LPREQ
LPSPI14 bus low power request.

enumerator kPower_LPSPI14_F_LPREQ
LPSPI14 function low power request.

enumerator kPower_LPSPI16_B_LPREQ
LPSPI16 bus low power request.

enumerator kPower_LPSPI16_F_LPREQ
LPSPI16 function low power request.

enumerator kPower_FLEXIO_B_LPREQ
FLEXIO bus low power request.

enumerator kPower_FLEXIO_F_LPREQ
FLEXIO function low power request.

enumerator kPower_NIC0_LPREQ
NIC0 low power request.

enumerator kPower_NIC1_LPREQ
NIC1 low power request.

enumerator kPower_ADC0_STOP
ADC0 stop request.

enumerator kPower_ACMP0_STOP
ACMP0 stop request.

enumerator kPower_MICFIL_STOP
MICFIL stop request.

enumerator kPower_LPI2C15_B_LPREQ
LPI2C15 bus low power request.

enumerator kPower_LPI2C15_F_LPREQ
LPI2C15 function low power request.

enumerator kPower_GDET3_LPREQ
GDET3 low power request.

enumerator kPower_RTC_STOP
RTC0 and RTC1 stop request.

enumerator kPower_EDMA2_STOP
EDMA2 stop request.

enumerator kPower_EDMA3_STOP
EDMA3 stop request.

enumerator kPower_FC17_B_LPREQ
LP_FLEXCOMM17 bus low power request.

enumerator kPower_FC18_B_LPREQ
LP_FLEXCOMM18 bus low power request.

enumerator kPower_FC19_B_LPREQ
LP_FLEXCOMM19 bus low power request.

enumerator kPower_FC20_B_LPREQ
LP_FLEXCOMM20 bus low power request.

enumerator kPower_FC17_F_LPREQ
LP_FLEXCOMM17 function low power request.

enumerator kPower_FC18_F_LPREQ
LP_FLEXCOMM18 function low power request.

enumerator kPower_FC19_F_LPREQ
LP_FLEXCOMM19 function low power request.

enumerator kPower_FC20_F_LPREQ
LP_FLEXCOMM20 function low power request.

enumerator kPower_SAI3_LPREQ
SAI3 low power request.

enumerator kPower_GPIO8_LPREQ
GPIO8 low power request.

enumerator kPower_GPIO9_LPREQ
GPIO9 low power request.

enumerator kPower_GPIO10_LPREQ
GPIO10 low power request.

enumerator kPower_I3C2_STOP
I3C2 low power request.

enumerator kPower_I3C3_STOP
I3C3 low power request.

enum __pmic_mode_reg
PMIC mode pin configuration API parameter.

Values:

enumerator kCfg_Run
PDRUNCFG0 register offset.

enumerator kCfg_Sleep
PDSLEEPCFG0 register offset.

enum __power_mode_config
Power mode configuration API parameter.

Values:

enumerator kPower_Sleep
Sleep mode.

enumerator kPower_DeepSleep
Deep Sleep mode.

enumerator kPower_DeepSleepRetention
Deep Sleep Retention(DSR) mode.

enumerator kPower_DeepPowerDown
Deep Power Down(DPD) mode.

enumerator kPower_FullDeepPowerDown
Full Deep Power Down(FDPD) mode.

enum _power_hwwake_src
DMA Wakeup sources.

Values:

enumerator kPower_HWWakeSrcFlexio
Enables DMA to wakeup by FLEXIO DMA request.

enumerator kPower_HWWakeSrcLpFlexcomm
Enables DMA to wakeup by LP_FLEXCOMM0-13(CPU0), LP_FLEXCOMM17-20(CPU1) DMA request.

enumerator kPower_HWWakeSrcSai
Enables DMA to wakeup by SAI0-2(CPU0), SAI3(CPU1) request.

enum _power_shared_resource_mask
Definition for shared resources.

Values:

enumerator kPower_MaskAdc0
ADC0 mask.

enumerator kPower_MaskAcmp0
ACMP0 mask.

enumerator kPower_MaskMicfil
MICFIL mask.

enumerator kPower_MaskLpi2c15
LPI2C15 mask.

enumerator kPower_MaskGdet3
GDET3 mask.

enumerator kPower_MaskRtc
RTC mask.

enumerator kPower_MaskVgpu
VGPU mask.

enumerator kPower_MaskEzhv
EZHV mask.

enumerator kPower_MaskXspi2
XSPI2 mask.

enumerator kPower_MaskLpspi14
LPSPI14 mask.

enumerator kPower_MaskLpspi16
LPSPI16 mask.

enumerator kPower_MaskFlexio
FlexIO mask.

enumerator kPower_MaskNic0
NIC0 mask.

enumerator kPower_MaskNic1
NIC1 mask.

enumerator kPower_MaskAll
Mask all shared modules.

enum _power_pad_vrange_val
pad voltage range value for the 3V capable IOs. To reduce power the IOs can be configured for 1.8V or 3.3V operation, powering down the detector, but requiring the application to restrict the IO supply voltage to the selected range.

Values:

enumerator kPadVol_Continuous
Continuous mode, VDDE detector on.

enumerator kPadVol_171_198
Voltage from 1.71V to 1.98V. supply detector off.

enumerator kPadVol_300_360
Voltage from 3.00V to 3.60V. supply detector off.

enum _body_bias_domain
Body Bias domain definition.

Values:

enumerator kPower_BodyBiasVdd2Sram
VDD2 domain RAMs.

enumerator kPower_BodyBiasVdd1Sram
VDD1 domain RAMs.

enumerator kPower_BodyBiasVddn
VDDN domain.

enumerator kPower_BodyBiasVdd2
VDD2 domain.

enumerator kPower_BodyBiasVdd1
VDD1 domain.

enum _power_regulator
Regulator definition for Regulator and LVD configuration.

Values:

enumerator kRegulator_DCDC
DCDC used for VDDN.

enumerator kRegulator_Vdd2LDO
VDD2 LDO.

enumerator kRegulator_Vdd1LDO
VDD1 LDO.

enum _power_ldo_mode
LDO mode definition.

Values:

enumerator kPower_LDOMode_Bypass
LDO Bypass mode.

enumerator kPower_LDOMode_HP
LDO High Power mode.

enumerator kPower_LDOMode_LP
LDO Low Power mode.

enum _power_dcdc_mode
DCDC mode definition.

Values:

enumerator kPower_DCDCMode_HP
DCDC High Power mode.

enumerator kPower_DCDCMode_LP
DCDC Low Power mode.

enumerator kPower_DCDCMode_ULP
DCDC Ultra Low Power mode. A0 Silicon does not support this mode and will default to low power mode when configured.

enum _power_vdd_src
VDDN, VDD1 or VDD2 supply source.

Values:

enumerator kVddSrc_PMC
Supplied by onchip regulator in PMC, DCDC for VDDN, LDO1 for VDD1, LDO2 for VDD2.

enumerator kVddSrc_PMIC
Supplied by external PMIC.

typedef enum *pd_bits* pd_bit_t
Power down control bits defined in SLEEPCON and PMC.

typedef enum *lp_bits* lp_bit_t
low power control bits for modules. Bits used for modules to requesting low-power.

typedef enum *_pmic_mode_reg* pmic_mode_reg_t
PMIC mode pin configuration API parameter.

typedef enum *_power_mode_config* power_mode_cfg_t
Power mode configuration API parameter.

typedef enum *_power_pad_vrange_val* power_pad_vrange_val_t
pad voltage range value for the 3V capable IOs. To reduce power the IOs can be configured for 1.8V or 3.3V operation, powering down the detector, but requiring the application to restrict the IO supply voltage to the selected range.

typedef enum *_power_regulator* power_regulator_t
Regulator definition for Regulator and LVD configuration.

typedef union *_power_regulator_voltage* power_regulator_voltage_t
Regulator voltage configuraton. Configure the regulator voltage in uV.

typedef enum *_power_vdd_src* power_vdd_src_t
VDDN, VDD1 or VDD2 supply source.

typedef union *_power_lvd_voltage* power_lvd_voltage_t
LVD voltage configuraton. Configure the LVD voltage in uV. Falling trip = 0.5V + 10 mV * bitfield value.

void POWER_EnablePD(*pd_bit_t* en)

API to enable PDRUNCFG bit in the SLEEPCON and PMC. Note that enabling the bit powers down the peripheral.

Parameters

- en – peripheral for which to enable the PDRUNCFG bit

void POWER_DisablePD(*pd_bit_t* en)

API to disable PDRUNCFG bit in the SLEEPCON and PMC. Note that disabling the bit powers up the peripheral.

Parameters

- en – peripheral for which to disable the PDRUNCFG bit

status_t POWER_ModuleEnterLPRequest(*lp_bit_t* en)

API to request a module entering low power. This API send low power/stop requests to a module and wait the acknowledgement from the module. For peripherals that are being used by a domain, using this low power request handshake is recommended to make sure the module is fully stopped before the module is powered down or clock gated.

Parameters

- en – peripheral for which to enable the low power request.

Returns

kStatus_Success for succeed, kStatus_Timeout for tiemout.

void POWER_ModuleExitLPRequest(*lp_bit_t* en)

API to negate low power/stop requests to a module(module resume regular operation).

Parameters

- en – peripheral for which to disable the low power request.

void POWER_SetPmicMode(uint32_t mode, *pmic_mode_reg_t* reg)

Set PMIC_MODE pins configure value.

Parameters

- mode – : PMIC MODE pin value. This field is aggregated low.
- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns

PMIC_MODE pins value in PDSLEEPCFG0

uint32_t POWER_GetPmicMode(*pmic_mode_reg_t* reg)

Get PMIC_MODE pins configure value.

Parameters

- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns

PMIC_MODE pins value in PDSLEEPCFG0 or PDRUNCFG0

static inline void POWER_EnableDeepSleep(void)

API to enable deep sleep bit in the ARM Core.

static inline void POWER_DisableDeepSleep(void)

API to disable deep sleep bit in the ARM Core.

void POWER_ApplyPD(void)

API to apply updated PMC PDRUNCFG bits in the Sysctl0.

void POWER_ClearEventFlags(uint32_t statusMask)

Clears the PMC event flags state.

Parameters

- statusMask – : A bitmask of event flags that are to be cleared.

uint32_t POWER_GetEventFlags(void)

Get the PMC event flags state.

Returns

PMC FLAGS register value

void POWER_EnableInterrupts(uint32_t interruptMask)

Enable the PMC interrupt requests.

Parameters

- interruptMask – : A bitmask of interrupts to enable.

void POWER_DisableInterrupts(uint32_t interruptMask)

Disable the PMC interrupt requests.

Parameters

- interruptMask – : A bitmask of interrupts to disable.

void POWER_EnableResets(uint32_t resetMask)

Enable the PMC reset requests.

Parameters

- resetMask – : A bitmask of Reset to enable, refer to PMC CTRL register descritpion in RM.

void POWER_DisableResets(uint32_t resetMask)

Disable the PMC reset requests.

Parameters

- resetMask – : A bitmask of Reset to disable, refer to PMC CTRL register descritpion in RM.

void EnableDeepSleepIRQ(IRQn_Type interrupt)

Enable specific interrupt for wake-up from deep-sleep mode. Enable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note: This function also enables the interrupt in the NVIC (EnableIRQ() is called internally).

Parameters

- interrupt – The IRQ number.

void DisableDeepSleepIRQ(IRQn_Type interrupt)

Disable specific interrupt for wake-up from deep-sleep mode. Disable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note: This function also disables the interrupt in the NVIC (DisableIRQ() is called internally).

Parameters

- interrupt – The IRQ number.

void POWER_EnableDMAHWWake(uint32_t sources)

Enable specific DMA channels to generate a wake-up from deep-sleep mode. DMA wakeup is an automatic hardware mechanism that handles DMA transactions for selected modules without the need for core processing. Software can configure individual DMA channels to generate a wakeup when they receive a DMA request. The DMA wakeup triggers an exit from Deep Sleep mode except the core, which remains sleeping.

Parameters

- sources – The bits mask for eDMA wakeup sources. Refers _power_hwwake_src.

void POWER_DisableDMAHWWake(uint32_t sources)

Disable specific DMA channels to generate a wake-up from deep-sleep mode. Disable the DMA channels for wake-up from deep sleep mode.

Parameters

- sources – The bits mask for eDMA wakeup sources. Refers _power_hwwake_src.

void POWER_EnableAutoWake(uint16_t ticks)

Configure PMC for auto wakeup.This feature allows PMC to wake up some amount of time after the domain enters Deep Sleep. The timer is clocked by PMC's internal 16 MHz clock. If both timers in compute and sense are enabled at the same time, wake-up will occur after BOTH timers time out (i.e. the max timeout value between the two is used).

Parameters

- ticks – : The ticks of 16MHz PMC internal clock to wait before waking up the domain after it goes to DeepSleep.

void POWER_EnableLPRequestMask(uint32_t mask)

Enable low power request masking for shared modules. Enabling the mask will relinquish the caller domain's control over the module, and give the other domain exclusive control over low-power requests to the module. NOTE, The masking of a shared module must NOT be disabled in both Compute and Sense domain.

Parameters

- mask – An OR'd bitmask of shared modules to be masked from low power request, ref _power_shared_resource_mask and SLEEPCON SHARED_MASK0 register in the RM. DeepSleep.

void POWER_DisableLPRequestMask(uint32_t mask)

Disable low power request masking for shared modules. Disabling the mask means this domain will own the module. Only one domain should own each shared module.

Parameters

- mask – An OR'd bitmask of shared modules to be enabled from low power request, ref _power_shared_resource_mask and SLEEPCON SHARED_MASK0 register in the RM. DeepSleep.

void POWER_EnableRunAFBB(uint32_t mask)

Enable AFBB mode for various domains in active mode. Note, users should call POWER_ApplyPD() to make the change take effect.

Parameters

- mask – : A bitmask of domains to enable AFBB mode, refer to _body_bias_domain and PMC PDRUNCFG0 register descritpion in RM.

void POWER_EnableSleepAFBB(uint32_t mask)

Enable AFBB mode for various domains in deep sleep mode. Note, users should call POWER_ApplyPD() to make the change take effect.

Parameters

- mask – : A bitmask of domains to enable AFBB mode, refer to _body_bias_domain and PMC PDSLEEPCFG0 register descritpion in RM.

void POWER_EnableRunRBB(uint32_t mask)

Enable RBB mode for various domains in active mode. This function should only be used when one CPU will not be making use of one of the voltage domains.Configuring the unused voltages for RBB will allow the other domain to have full control of the body bias mode for the voltages even when both domains are in run mode. Note, users should call POWER_ApplyPD() to make the change take effect.

Parameters

- mask – : A bitmask of domains to enable RBB mode, refer to _body_bias_domain and PMC PDRUNCFG0 register descritpion in RM.

void POWER_EnableSleepRBB(uint32_t mask)

Enable RBB mode for various domains in deep sleep mode.

Parameters

- mask – : A bitmask of domains to enable RBB mode, refer to _body_bias_domain and PMC PDSLEEPCFG0 register descritpion in RM.

void POWER_EnableRunNBB(uint32_t mask)

Enable NBB mode for various domains in active mode. NBB can be a lower power option than RBB if the domain enters and exits low-power modes frequently since the wells don't charge and discharge as much on every entry/exit.

Parameters

- mask – : A bitmask of domains to enable NBB mode, refer to _body_bias_domain and PMC PDRUNCFG0 register descritpion in RM.

void POWER_EnableSleepNBB(uint32_t mask)

Enable NBB mode for various domains in deep sleep mode. NBB can be a lower power option than RBB if the domain enters and exits low-power modes frequently since the wells don't charge and discharge as much on every entry/exit.

Parameters

- mask – : A bitmask of domains to enable NBB mode, refer to _body_bias_domain and PMC PDSLEEPCFG0 register descritpion in RM.

uint32_t POWER_CalcVoltLevel(*power_regulator_t* regulator, uint32_t maxFreqHz, uint32_t miniVoltUV)

Get the minimum operative(active) voltage for VDD1 or VDD2 with the given maximum main clock frequency in the domain.

Parameters

- maxFreqHz – the maxiumum main clock frequency in Hz will be used for the domain.

- miniVoltUV – the minimum voltage in microvolt(uV) for VDD1 or VDD2. Should <= 1100000uV, 0 means use the main clock frequency to calculate voltage.

Returns

the calculated operative voltage in uV. 0xFFFFFFFFU means exceeded max supported value.

status_t POWER_ConfigRegulatorSetpointsForFreq(*power_regulator_t* regulator, uint32_t *maxFreqHz, uint32_t *miniVoltUV, uint32_t startPoint, uint32_t num)

Configure the setpoint operation for on-chip regulators(LDO1/LDO2) and LVD. This function can configure both the target output voltages and LVD to suitable values per the desired frequency for the specific setpoints. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL]. Use the POWER_SelectRunSetpoint() or POWER_SelectSleepSetpoint() to configure the setpoint selections. The voltage selected for each regulator must also be in ascending order (VSEL0 <= VSEL1 <= VSEL2 <= VSEL3).

Parameters

- maxFreqHz – the maxiumum main clock frequency in Hz will be used for the domain. 0 means use the miniVolt defined value and can be used to configure setpoint for DeepSleep.
- miniVolt – the minimum voltage in microvolt(uV) for VDD1 or VDD2. Should <= 1100000uV, 0 means use the main clock frequency to calculate voltage.
- startPoint – the start setpoint to be configured.
- num – the count of setpoints to be configured, the startPoint+num should not larger than 4.

Returns

kStatus_Success for successful configuration, kStatus_InvalidArgument for wrong parameter.

status_t POWER_ConfigRegulatorSetpoints(*power_regulator_t* regulator, const *power_regulator_voltage_t* *volt, const *power_lvd_voltage_t* *lvd)

Configure the setpoint operation for on-chip regulators and LVD. This function can configure both the target output voltages and LVD levels for each setpoint. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL]. Use the POWER_SelectRunSetpoint() or POWER_SelectSleepSetpoint() to configure the setpoint selections. The voltages selected for each regulator must also be in ascending order (VSEL0 <= VSEL1 <= VSEL2 <= VSEL3). The voltages selected for each LVD must also be in ascending order (LVL0 <= LVL1 <= LVL2 <= LVL3).

NOTE, Only valid regulator for sense domain is VDD1 LDO. Note, LDOVDD1VSEL and LVDVDD1CTRL are shared between the compute and sense domains. There is no aggregation to combine the values. Whichever domain wrote the register last determines the value. It is recommended that only one domain be used to configure VDD1 setpoints.

```
const power_regulator_voltage_t regulator = {
  .LDO.vsel0 = 700000,
  .LDO.vsel1 = 800000,
  .LDO.vsel2 = 900000,
  .LDO.vsel3 = 1000000,
};
```

(continues on next page)

(continued from previous page)

```
const power_lvd_voltage_t lvd = {
  .VDD12.lvl0 = 600000,
  .VDD12.lvl1 = 700000,
  .VDD12.lvl2 = 800000,
  .VDD12.lvl3 = 900000,
};

ret = POWER_ConfigRegulatorSetpoints(kRegulator_Vdd1LDO, &regulator, &lvd);
```

Parameters

- regulator – which regulator or power domain to configure, refer to power_regulator_t.
- volt – regulator configuration pointer, refer to power_regulator_voltage_t. Note, only two setpoints are available for DCDC.
- lvd – LVD voltage configuration, refer to power_lvd_voltage_t.

Returns

kStatus_Success for succeed, kStatus_InvalidArgument for wrong arguments.

status_t POWER_ConfigLvdSetpoints(*power_regulator_t* regulator, const *power_lvd_voltage_t* *lvd)

Configure the setpoint operation for LVD. This function can configure LVD levels for each setpoint. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL]. Use the POWER_SelectRunSetpoint() or POWER_SelectSleepSetpoint() to configure the setpoint selections. The voltages selected for each LVD must be in ascending order (LVL0 <= LVL1 <= LVL2 <= LVL3).

NOTE, Only valid regulator for sense domain is VDD1 LDO. Note, LDOVDD1VSEL and LDVDD1CTRL are shared between the compute and sense domains. There is no aggregation to combine the values. Whichever domain wrote the register last determines the value. It is recommended that only one domain be used to configure VDD1 setpoints.

```
const power_lvd_voltage_t lvd = {
  .VDD12.lvl0 = 600000,
  .VDD12.lvl1 = 700000,
  .VDD12.lvl2 = 800000,
  .VDD12.lvl3 = 900000,
};

ret = POWER_ConfigLvdSetpoints(kRegulator_Vdd1LDO, &lvd);
```

Parameters

- regulator – which regulator or power domain to configure, refer to power_regulator_t.
- lvd – LVD voltage configuration, refer to power_lvd_voltage_t.

Returns

kStatus_Success for succeed, kStatus_InvalidArgument for wrong arguments.

void POWER_GetLvdSetpoints(*power_regulator_t* regulator, *power_lvd_voltage_t* *lvd)

Get the LVD setpoints configurations. This function can get the LVD levels for each setpoint. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL].

NOTE, Only valid regulator for sense domain is VDD1 LDO.

Parameters

- regulator – which regulator or power domain to get, refer to power_regulator_t.
- lvd – LVD voltage configuration, refer to power_lvd_voltage_t.

void POWER_SelectRunSetpoint(*power_regulator_t* regulator, uint32_t setpoint)

Select the setpoint(target voltage and LVD threshold) for active mode. The requests from the compute and sense domains are aggregated, so depending on the other domain's configuration the setpoint might not change or might change later (when the other domain also requests a setpoint change). Note, users should call POWER_ApplyPD() to make the change take effect.

Parameters

- regulator – which regulator or power domain to configure, refer to power_regulator_t.
- setpoint – select which setpoint is used. Valid options for DCDC are 0-1. Valid options for VDD1/VDD2 are 0-3.

void POWER_SelectSleepSetpoint(*power_regulator_t* regulator, uint32_t setpoint)

Select setpoint(target voltage and LVD threshold) in Deep Sleep (or lower) mode. Use POWER_ConfigRegulatorSetpoints() to change the target voltage and/or LVD threshold value corresponding to each setpoint. The requests from the compute and sense domains are aggregated,.

Parameters

- regulator – which regulator or power domain to configure, refer to power_regulator_t.
- setpoint – select which setpoint is used. Valid options for DCDC are 0-1. Valid options for VDD1/VDD2 are 0-3.

void POWER_SetRunRegulatorMode(*power_regulator_t* regulator, uint32_t mode)

Set the on-chip regulator mode in Run/Active mode. The requests from the compute and sense domains are aggregated.

Parameters

- regulator – which regulator or power domain to configure, refer to power_regulator_t.
- mode – set the workmode. Refer to _power_dcdc_mode for available DCDC modes, refer to _power_ldo_mode for LDO available modes.

void POWER_SetSleepRegulatorMode(*power_regulator_t* regulator, uint32_t mode)

Set the on-chip regulator mode in Deep Sleep mode. The requests from the compute and sense domains are aggregated.

Parameters

- regulator – which regulator or power domain to configure, refer to power_regulator_t.
- mode – set the workmode. Refer to _power_dcdc_mode for available DCDC modes, refer to _power_ldo_mode for LDO available modes.

void POWER_ResetIOBank(uint32_t mask)

Reset the IO bank. Sets the IO bank reset which tristates the ports. Intended to be used prior to powering off the VDDIO supplies.

Parameters

- mask – A bitmask of IO Banks to be reseted, power_io_bank_t and PMC PADCFG register. The register reset by cold reset.

void POWER_IOBankIsolationHold(uint32_t mask)

IO Bank Isolation Hold. Maintains IO bank isolation state after wake-up from FDSR/DPD modes. If clear, the IO bank's state will be retained during FDSR & DPD modes, but will be controllable after wake-up. If set, the IO bank will remain in retain mode until ISOCTRL cleared by software.

Parameters

- mask – A bitmask of IO Banks to be set, power_io_bank_t and PMC PADCFG register. The register reset by cold reset.

void POWER_IOBankClearIsolationHold(uint32_t mask)

Clear the IO bank Isolation Hold and regain state control. This bit cannot be cleared if associated domain is not powered.

Parameters

- mask – A bitmask of IO Banks to be clear, power_io_bank_t and PMC PADCFG register. The register reset by cold reset.

void POWER_EnterSleep(void)

Configures and enters in SLEEP low power mode.

void POWER_EnterDeepSleep(const uint32_t exclude_from_pd[7])

PMC Deep Sleep function call.

NOTE, the body bias and DCDC, LDO configurations in PMC_PDSLEECFG0 is not covered by this API, please use POWER_EnableSleepRBB or POWER_EnableSleepNBB and POWER_SelectSleepSetpoint for those settings.

NOTE, the MAINPLL and AUDIO PLL need special power up sequence, so it can't be power down/on automatically during the mode change. Application need turn off the PLL before the low power mode entry and re-initialize the PLL after wakeup.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during Deep Sleep mode. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

void POWER_RequestDSR(const uint32_t exclude_from_pd[7])

Sense domain requests entering Deep Sleep Retention mode API. Deep Sleep Retention (DSR) mode is a deeper low power state than Deep Sleep mode. DSR mode allows significant portions of the chip to be powered-down while maintaining some level of state retention. The Sense domain can enter DSR mode only when all other domains are in DSR mode. If the condition is false, the sense domain will be in a shallower power mode(Deep sleep mode).

NOTE, the body bias and DCDC, LDO configurations in PMC_PDSLEECFG0 is not covered by this API, please use POWER_EnableSleepRBB or POWER_EnableSleepNBB and POWER_SelectSleepSetpoint for those settings.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on(bits cleared) during the power mode. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

void POWER_RequestDeepPowerDown(const uint32_t exclude_from_pd[7])

The domain requests to enter Deep Power Down mode. The Deep Power Down and Full Deep Power Down can occur only when both domains signal their intent to enter these

lower power modes. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during Deep Power Down.

void POWER_RequestFullDeepPowerDown(const uint32_t exclude_from_pd[7])

The domain requests to enter Full Deep Power Down mode. The Deep Power Down and Full Deep Power Down can occur only when both domains signal their intent to enter these lower power modes. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during Deep Power Down.

void POWER_EnterPowerMode(*power_mode_cfg_t* mode, const uint32_t exclude_from_pd[7])

Power Library API to request entering different power mode. This API is used for requesting entering the target mode, the final SOC power mode depends on hardware aggregation. Note, the Sense domain can enter DSR mode only when Compute domain is in DSR. The SOC enters DPD or FDPD when both domain requested entering DPD or FDPD.

Parameters

- mode – Power mode to enter.
- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during power mode selected.

uint32_t POWER_GetLibVersion(void)

Power Library API to return the library version.

Returns

version number of the power library

FSL_POWER_DRIVER_VERSION

power driver version 2.5.0.

POWER_DEFAULT_PMICMODE_DS

Define the default PMIC modes for power modes.

POWER_DEFAULT_PMICMODE_DSR

POWER_DEFAULT_PMICMODE_DPD

POWER_DEFAULT_PMICMODE_FDPD

POWER_DEFAULT_LVD_VOLT

Define the default LVD threshold.

POWER_LDO_SAFE_MARGIN(x)

Define the safe margin for regulator output voltage levels.

MAKE_PD_BITS(reg, slot)

GET_PD_REG_FROM_BITS(bit)

SLEEPCON_RCFG

PMC_PDRCFG0

PMC_PDRCFG1

PMC_PDRCFG2

PMC_PDRCFG3

PMC_PDRCFG4

PMC_PDRCFG5

SHA_MED_CCTRL0_OFFSET

SHA_MED_TCTRL0_OFFSET

SHA_SEN_TCTRL0_OFFSET

PRIVATE_CCTRL0_OFFSET

PRIVATE_TCTRL0_OFFSET

PRIVATE_TCTRL1_OFFSET

PRIVATE_TCTRL2_OFFSET

SHA_MED_CSTAT0_OFFSET

SHA_MEDSEN_TSTAT0_OFFSET

PRIVATE_CSTAT0_OFFSET

PRIVATE_TSTAT0_OFFSET

PRIVATE_TSTAT1_OFFSET

POWER_LP_REQ_CTRL_REG_OFFSET

POWER_LP_REQ_STATE_REG_OFFSET

POWER_LP_REQ_STATE_BIT_OFFSET

POWER_MAKE_LP_BITS(ctrl_reg, ctrl_bit, state_reg, state_bit)

GET_LP_CTRL_REG_FROM_BITS(bit)

GET_LP_CTRL_BIT_FROM_BITS(bit)

GET_LP_STATE_REG_FROM_BITS(bit)

GET_LP_STATE_BIT_FROM_BITS(bit)

uint32_t vsel0
: DCDC VSEL0. Voltage = 0.5 V + 6.25 mV * value of vsel0. Maximum value = 1150000uV.

uint32_t vsel1
: DCDC VSEL1. Voltage = 0.5 V + 6.25 mV * value of vsel1. Maximum value = 1150000uV.

struct *_power_regulator_voltage* DCDC

uint32_t vsel0
: LDO VSEL0. Voltage = 0.45 V + 12.5 mV * vsel0. Maximum value = 1150000uV.

uint32_t vsel1
: LDO VSEL1. Voltage = 0.45 V + 12.5 mV * vsel1. Maximum value = 1150000uV.

uint32_t vsel2
: LDO VSEL2. Voltage = 0.45 V + 12.5 mV * vsel2. Maximum value = 1150000uV.

uint32_t vsel3
: LDO VSEL3. Voltage = 0.45 V + 12.5 mV * vsel3. Maximum value = 1150000uV.

struct *_power_regulator_voltage* LDO

uint32_t vsel[4]

uint32_t lvl0
: VDDN LVD level0, uV.

uint32_t lvl1
: VDDN LVD level1, uV.

struct *_power_lvd_voltage* VDDN

uint32_t lvl0
: VDD1/2 LVD level0, uV.

uint32_t lvl1
: VDD1/2 LVD level1, uV.

uint32_t lvl2
: VDD1/2 LVD level2, uV.

uint32_t lvl3
: VDD1/2 LVD level3, uV.

struct *_power_lvd_voltage* VDD12

uint32_t lvl[4]

union __power__regulator__voltage
: *#include <fsl_power.h>* Regulator voltage configuraton. Configure the regulator voltage in uV.

union __power__lvd__voltage
: *#include <fsl_power.h>* LVD voltage configuraton. Configure the LVD voltage in uV. Falling trip = 0.5V + 10 mV * bitfield value.

struct DCDC

Public Members

uint32_t vsel0
: DCDC VSEL0. Voltage = 0.5 V + 6.25 mV * value of vsel0. Maximum value = 1150000uV.

uint32_t vsel1
: DCDC VSEL1. Voltage = 0.5 V + 6.25 mV * value of vsel1. Maximum value = 1150000uV.

struct LDO

Public Members

uint32_t vsel0
: LDO VSEL0. Voltage = 0.45 V + 12.5 mV * vsel0. Maximum value = 1150000uV.

uint32_t vsel1
: LDO VSEL1. Voltage = 0.45 V + 12.5 mV * vsel1. Maximum value = 1150000uV.

uint32_t vsel2
: LDO VSEL2. Voltage = 0.45 V + 12.5 mV * vsel2. Maximum value = 1150000uV.

uint32_t vsel3
: LDO VSEL3. Voltage = 0.45 V + 12.5 mV * vsel3. Maximum value = 1150000uV.

struct VDDN

Public Members

uint32_t lvl0
: VDDN LVD level0, uV.

uint32_t lvl1
: VDDN LVD level1, uV.

struct VDD12

Public Members

uint32_t lvl0
: VDD1/2 LVD level0, uV.

uint32_t lvl1
: VDD1/2 LVD level1, uV.

uint32_t lvl2
: VDD1/2 LVD level2, uV.

uint32_t lvl3
: VDD1/2 LVD level3, uV.

2.70 PUF: Physical Unclonable Function

2.71 Puf_v3_driver

FSL_PUF_V3_DRIVER_VERSION
: PUFv3 driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- 2.0.4
  - Fix warning
- 2.0.3
  - Update for various PUF CTRL wrapper
- 2.0.2
  - Fix MISRA issue in driver.
- 2.0.1
  - Fix PUF initialization issue and update driver to reflect SoC header changes.
- 2.0.0

– Initial version.

Values:

enumerator kStatus_PUF_OperationNotAllowed

enumerator kStatus_PUF_AcNotForThisProductPhase1

enumerator kStatus_PUF_AcNotForThisProductPhase2

enumerator kStatus_PUF_AcCorruptedPhase1

enumerator kStatus_PUF_AcCorruptedPhase2

enumerator kStatus_PUF_AcAuthFailedPhase1

enumerator kStatus_PUF_NBOOT_AcAuthFailedPhase2

enumerator kStatus_PUF_QualityVerificationFail

enumerator kStatus_PUF_ContextIncorrect

enumerator kStatus_PUF_DestinationNotAllowed

enumerator kStatus_PUF_Failure

typedef uint32_t puf_endianness_t

typedef uint32_t puf_key_dest_t

typedef uint32_t puf_key_scope_t

typedef uint32_t puf_result_code_t

typedef uint32_t puf_sec_level_t

puf_endianness_t dataEndianness

uint8_t CKGATING

puf_key_scope_t keyScopeStarted

puf_key_scope_t keyScopeEnrolled

uint32_t userCtx0

uint32_t userCtx1

void PUF_GetDefaultConfig(*puf_config_t* *conf)

brief Sets the default configuration of PUF

This function initialize PUF config structure to default values.

Parameters

- conf – PUF configuration structure

status_t PUF_Init(PUF_Type *base, *puf_config_t* *conf)

brief Initialize PUF

This function enables power to PUF block and waits until the block initializes.

Parameters

- conf – PUF configuration structure

Returns

Status of the init operation

void PUF_Deinit(PUF_Type *base, *puf_config_t* *conf)

brief Denitialize PUF

This function disables power to PUF SRAM and peripheral clock.

Parameters

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

status_t PUF_Enroll(PUF_Type *base, uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)

brief Enroll PUF

This function derives a digital fingerprint, generates the corresponding Activation Code (AC) and returns it to be stored in an NVM or a file. This step needs to be performed only once for each device. This function may be permanently disallowed by a fuse.

Parameters

- base – PUF peripheral base address
- activationCode – **[out]** Word aligned address of the resulting activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns

Status of enroll operation.

status_t PUF_Start(PUF_Type *base, const uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)

brief Start PUF

The Activation Code generated during the Enroll operation is used to reconstruct the digital fingerprint. This needs to be done after every power-up and reset.

Parameters

- base – PUF peripheral base address
- activationCode – **[in]** Word aligned address of the input activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the start operation. return Status of start operation.

status_t PUF_Stop(PUF_Type *base)

brief Stop PUF

The Stop operation removes all key material from PUF flipflops and PUF SRAM, and sets PUF to the Stopped state.

Parameters

- base – PUF peripheral base address

Returns

Status of stop operation.

status_t PUF_GetKey(PUF_Type *base, *puf_key_ctx_t* *keyCtx, *puf_key_dest_t* keyDest, uint8_t *key, size_t keySize)

brief PUF Get Key

The Get Key operation derives a key from the intrinsic PUF key and externally provided context.

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keyDest – output destination of the derived PUF key
- key – **[out]** Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the derived key in bytes.

Returns

Status of get key operation.

status_t PUF_WrapGeneratedRandom(PUF_Type *base, *puf_key_ctx_t* *keyCtx, size_t keySize, uint8_t *keyCode, size_t keyCodeSize)

brief PUF Wrap generated random

The Wrap Generated Random operation wraps a random key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keySize – Size of the key to be generated in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns

Status of wrap generated random operation.

status_t PUF_Wrap(PUF_Type *base, *puf_key_ctx_t* *keyCtx, uint8_t *userKey, size_t userKeySize, uint8_t *keyCode, size_t keyCodeSize)

brief PUF Wrap user key

The Wrap operation wraps a user defined key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct.
- userKey – Word aligned address of input user key.
- userKeySize – Size of the key to be wrapped in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns

Status of wrap operation.

status_t PUF_Unwrap(PUF_Type *base, *puf_key_dest_t* keyDest, uint8_t *keyCode, size_t keyCodeSize, uint8_t *key, size_t keySize)

brief PUF Unwrap user key

The unwrap operation unwraps the key from a previously created Key Code (KC)

Parameters

- base – PUF peripheral base address
- keyDest – output destination of the unwraped PUF key
- keyCode – **[in]** Word aligned address of the input key code.
- keyCodeSize – Size of the input keycode in bytes.
- key – Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the key to be generated in bytes.

Returns

Status of unwrap operation.

status_t PUF_GenerateRandom(PUF_Type *base, uint8_t *data, size_t size)

brief Generate Random

The Generate Random operation outputs the requested amount of random data as specified in a provided context.

Parameters

- base – PUF peripheral base address
- size – Size of random data to be genarated in bytes.

Returns

Status of generate random operation.

status_t PUF_Zeroize(PUF_Type *base)

brief Zeroize PUF

This function clears all PUF internal logic and puts the PUF to zeroized state.

Parameters

- base – PUF peripheral base address

Returns

Status of the zeroize operation.

status_t PUF_Test(PUF_Type *base, uint8_t *score)

brief Test PUF

With the Test PUF operation, diagnostics about the PUF quality is collected and presented in a PUF score.

Parameters

- base – PUF peripheral base address
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns

Status of the test operation.

static inline void PUF_BlockCommand(PUF_Type *base, uint32_t mask)

Blocks specified PUF commands.

This function blocks PUF commands specified by mask parameter.

Parameters

- base – PUF peripheral base address
- mask – Mask of parameters which should be blocked until power-cycle.

Returns

Status of the test operation.

status_t PUF_SetLock(PUF_Type *base, *puf_sec_level_t* securityLevel)

brief Set lock of PUF operation

Lock the security level of PUF block until key generate, wrap or unwrap operation is completed. Note: Only security level defined in SEC_LOCK register can use PUFv3 or change its security level. Default setting after leaving ROM is Secure-Privilege

Parameters

- base – PUF peripheral base address
- securityLevel – Security level of PUF block.

Returns

Status of the test operation.

status_t PUF_SetCtxMask(PUF_Type *base, uint32_t appCtxMask)

brief Set App Context mask

This function sets Application defined context mask used in conjunction with key user context 2. Whenever bit in this register is 1, corresponding bit in user context 2 provided during key code creation should be zero only.

This register is only modifiable by task running at secure-privilege level.

Parameters

- base – PUF peripheral base address
- appCtxMask – Value of the Application defined context mask.

Returns

Status of the test operation.

kPUF_EndianLittle

kPUF_EndianBig

kPUF_KeyDestRegister

kPUF_KeyDestKeyBus

kPUF_KeyDestInvalid

kPUF_KeyAllowRegister

kPUF_KeyAllowKeyBus

kPUF_KeyAllowAll

kPUF_ResultOK

kPUF_AcNotForThisProductPhase1

kPUF_AcNotForThisProductPhase2

kPUF_AcCorruptedPhase1

kPUF_AcCorruptedPhase2

kPUF_AcAuthFailedPhase1

kPUF_AcAuthFailedPhase2

kPUF_QualityVerificationFail

kPUF_ContextIncorrect

kPUF_DestinationNotAllowed

kPUF_Failure

kPUF_NonsecureUser

kPUF_NonsecurePrivilege

kPUF_SecureUser

kPUF_SecurePrivilege

PUF_ACTIVATION_CODE_SIZE

PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(**x**)

SEC_LOCK_PATTERN

struct puf_config_t
#include <fsl_puf_v3.h>

struct puf_key_ctx_t
#include <fsl_puf_v3.h>

2.72 Reset Driver

enum _rstctl_reset_source
Enumeration for system reset status bits.
Defines the enumeration for system reset status bits in SYSRSTSTAT register
Values:
enumerator kRSTCTL_SourceVddPor
VDD Power-On Reset(POR)
enumerator kRSTCTL_SourcePad
PPAD Reset
enumerator kRSTCTL_SourceIspAp
ISP_AP reset Reset
enumerator kRSTCTL_SourceItrcSw
ITRC_SW (Intrusion and Tamper Response Controller SW) Reset
enumerator kRSTCTL_SourceCpu0
VDD2_COMP Core Reset
enumerator kRSTCTL_SourceCpu1
VDD1_SENSE Core Reset

enumerator kRSTCTL_SourceWwdt0
WatchDog Timer 0 Reset

enumerator kRSTCTL_SourceWwdt1
WatchDog Timer 1 Reset

enumerator kRSTCTL_SourceWwdt2
WatchDog Timer 2 Reset

enumerator kRSTCTL_SourceWwdt3
WatchDog Timer 3 Reset

enumerator kRSTCTL_SourceCdog0
Code WatchDog Timer 0 Reset

enumerator kRSTCTL_SourceCdog1
Code WatchDog Timer 1 Reset

enumerator kRSTCTL_SourceCdog2
Code WatchDog Timer 2 Reset

enumerator kRSTCTL_SourceCdog3
Code WatchDog Timer 3 Reset

enumerator kRSTCTL_SourceCdog4
Code WatchDog Timer 4 Reset

enumerator kRSTCTL_SourceAll

enum _RSTCTL_RSTn
Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in RSTCLTx registers

Values:

enumerator kIOPCTL0_RST_SHIFT_RSTn
IOPCTL0 reset control

enumerator kELS_RST_SHIFT_RSTn
ELS S50 reset control

enumerator kDMA0_RST_SHIFT_RSTn
eDMA0 reset control

enumerator kDMA1_RST_SHIFT_RSTn
eDMA1 reset control

enumerator kPKC_RST_SHIFT_RSTn
PKC reset control

enumerator kXSPI0_RST_SHIFT_RSTn
XSPI0 reset control

enumerator kXSPI1_RST_SHIFT_RSTn
XSPI1 reset control

enumerator kGPIO0_RST_SHIFT_RSTn
GPIO0 reset control

enumerator kGPIO1_RST_SHIFT_RSTn
GPIO1 reset control

enumerator kGPIO2_RST_SHIFT_RSTn
GPIO2 reset control

enumerator kGPIO3_RST_SHIFT_RSTn
GPIO3 reset control

enumerator kGPIO4_RST_SHIFT_RSTn
GPIO4 reset control

enumerator kGPIO5_RST_SHIFT_RSTn
GPIO5 reset control

enumerator kGPIO6_RST_SHIFT_RSTn
GPIO6 reset control

enumerator kGPIO7_RST_SHIFT_RSTn
GPIO7 reset control

enumerator kSCT0_RST_SHIFT_RSTn
SCT0 reset control

enumerator kFC0_RST_SHIFT_RSTn
LP_FLEXCOMM0 reset control

enumerator kFC1_RST_SHIFT_RSTn
LP_FLEXCOMM1 reset control

enumerator kFC2_RST_SHIFT_RSTn
LP_FLEXCOMM2 reset control

enumerator kFC3_RST_SHIFT_RSTn
LP_FLEXCOMM3 reset control

enumerator kFC4_RST_SHIFT_RSTn
LP_FLEXCOMM4 reset control

enumerator kFC5_RST_SHIFT_RSTn
LP_FLEXCOMM5 reset control

enumerator kFC6_RST_SHIFT_RSTn
LP_FLEXCOMM6 reset control

enumerator kFC7_RST_SHIFT_RSTn
LP_FLEXCOMM7 reset control

enumerator kFC8_RST_SHIFT_RSTn
LP_FLEXCOMM8 reset control

enumerator kFC9_RST_SHIFT_RSTn
LP_FLEXCOMM9 reset control

enumerator kFC10_RST_SHIFT_RSTn
LP_FLEXCOMM10 reset control

enumerator kFC11_RST_SHIFT_RSTn
LP_FLEXCOMM11 reset control

enumerator kFC12_RST_SHIFT_RSTn
LP_FLEXCOMM12 reset control

enumerator kFC13_RST_SHIFT_RSTn
LP_FLEXCOMM13 reset control

enumerator kSAI0_RST_SHIFT_RSTn
SAI0 reset control

enumerator kSAI1_RST_SHIFT_RSTn
SAI1 reset control

enumerator kSAI2_RST_SHIFT_RSTn
SAI2 reset control

enumerator kI3C0_RST_SHIFT_RSTn
I3C0 reset control

enumerator kI3C1_RST_SHIFT_RSTn
I3C1 reset control

enumerator kCRC0_RST_SHIFT_RSTn
CRC0 reset control

enumerator kCTIMER0_RST_SHIFT_RSTn
CTIMER0 reset control

enumerator kCTIMER1_RST_SHIFT_RSTn
CTIMER1 reset control

enumerator kCTIMER2_RST_SHIFT_RSTn
CTIMER2 reset control

enumerator kCTIMER3_RST_SHIFT_RSTn
CTIMER3 reset control

enumerator kCTIMER4_RST_SHIFT_RSTn
CTIMER4 reset control

enumerator kMRT0_RST_SHIFT_RSTn
MRT0 reset control

enumerator kUTICK0_RST_SHIFT_RSTn
UTICK0 reset control

enumerator kSEMA424_RST_SHIFT_RSTn
SEMA42_4 reset control

enumerator kMU4_RST_SHIFT_RSTn
MU4 reset control

enumerator kSYSPM2_RST_SHIFT_RSTn
SYSPM_XSPI0 reset control

enumerator kSYSPM3_RST_SHIFT_RSTn
SYSPM_XSPI1 reset control

enumerator kSAFO_SGI_RST_SHIFT_RSTn
SAFO_SGI reset control

enumerator kINPUTMUX0_RST_SHIFT_RSTn
INPUTMUX0 reset control

enumerator kFREQME_RST_SHIFT_RSTn
FREQME0 reset control

enumerator kSYSPM0_RST_SHIFT_RSTn
SYSPM_PC reset control

enumerator kSYSPM1_RST_SHIFT_RSTn
SYSPM_PS reset control

enumerator kNPU0_RST_SHIFT_RSTn
NPU0 reset control

enumerator kHIFI4_RST_SHIFT_RSTn
HiFi4 reset control

enumerator kHIFI4_DEBUG_RST_SHIFT_RSTn
HiFi4 Debug reset control

enumerator kHIFI1_RST_SHIFT_RSTn
HiFi1 reset control

enumerator kHIFI1_DEBUG_RST_SHIFT_RSTn
HiFi1 Debug reset control

enumerator kDMA2_RST_SHIFT_RSTn
eDMA2 reset control

enumerator kDMA3_RST_SHIFT_RSTn
eDMA3 reset control

enumerator kFC17_RST_SHIFT_RSTn
LP_FLEXCOMM17 reset control

enumerator kFC18_RST_SHIFT_RSTn
LP_FLEXCOMM18 reset control

enumerator kFC19_RST_SHIFT_RSTn
LP_FLEXCOMM19 reset control

enumerator kFC20_RST_SHIFT_RSTn
LP_FLEXCOMM20 reset control

enumerator kSAI3_RST_SHIFT_RSTn
SAI3 reset control

enumerator kI3C2_RST_SHIFT_RSTn
I3C2 reset control

enumerator kI3C3_RST_SHIFT_RSTn
I3C3 reset control

enumerator kGPIO8_RST_SHIFT_RSTn
GPIO8 reset control

enumerator kGPIO9_RST_SHIFT_RSTn
GPIO9 reset control

enumerator kGPIO10_RST_SHIFT_RSTn
GPIO10 reset control

enumerator kCTIMER5_RST_SHIFT_RSTn
CTIMER5 reset control

enumerator kCTIMER6_RST_SHIFT_RSTn
CTIMER6 reset control

enumerator kCTIMER7_RST_SHIFT_RSTn
CTIMER7 reset control

enumerator kMRT1_RST_SHIFT_RSTn
MRT1 reset control

enumerator kUTICK1_RST_SHIFT_RSTn
UTICK1 reset control

enumerator kMU3_RST_SHIFT_RSTn
MU3 reset control

enumerator kSEMA423_RST_SHIFT_RSTn
SEMA42_3 reset control

enumerator kPVTS1_RST_SHIFT_RSTn
PVTS1 reset control

enumerator kIOPCTL2_RST_SHIFT_RSTn
IOPCTL2 reset control

enumerator kIOPCTL1_RST_SHIFT_RSTn
IOPCTL1 reset control

enumerator kCPU1_RST_SHIFT_RSTn
CPU1 reset control

enumerator kMU0_RST_SHIFT_RSTn
MU0 reset control

enumerator kMU1_RST_SHIFT_RSTn
MU1 reset control

enumerator kMU2_RST_SHIFT_RSTn
MU2 reset control

enumerator kSEMA420_RST_SHIFT_RSTn
SEMA42_0 reset control

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control

enumerator kSDADC0_RST_SHIFT_RSTn
SDADC0 reset control

enumerator kACMP0_RST_SHIFT_RSTn
ACMP0 reset control

enumerator kPDM_RST_SHIFT_RSTn
MICFIL/PDM reset control

enumerator kINPUTMUX1_RST_SHIFT_RSTn
INPUTMUX1 reset control

enumerator kLPI2C15_RST_SHIFT_RSTn
LPI2C15 reset control

enumerator kVGPU_RST_SHIFT_RSTn
VGPU reset control

enumerator kLCDIF_RST_SHIFT_RSTn
LCDIF reset control

enumerator kMIPI_DSI_CTRL_RST_SHIFT_RSTn
MIPIDSI reset control

enumerator kEZHV_RST_SHIFT_RSTn
EZHV reset control

enumerator kJPEGDEC_RST_SHIFT_RSTn
JPEGDEC reset control

enumerator kPNGDEC_RST_SHIFT_RSTn
PNGDEC reset control

enumerator kXSPI2_RST_SHIFT_RSTn
XSPI2 and MMU2 reset control

enumerator kLPSPI14_RST_SHIFT_RSTn
LPSPI14 reset control

enumerator kLPSPI16_RST_SHIFT_RSTn
LPSPI16 reset control

enumerator kFLEXIO0_RST_SHIFT_RSTn
FLEXIO0 reset control

enumerator kUSB0_RST_SHIFT_RSTn
USB0 reset control

enumerator kUSBPHY0_RST_SHIFT_RSTn
USBPHY0 reset control

enumerator kUSB1_RST_SHIFT_RSTn
USB1 reset control

enumerator kUSDHC0_RST_SHIFT_RSTn
USDHC0 reset control

enumerator kUSDHC1_RST_SHIFT_RSTn
USDHC1 reset control

Flash device current running mode.

Values:

enumerator kFlashInstMode_ExtendedSpi

enumerator kFlashInstMode_0_4_4_SDR

enumerator kFlashInstMode_0_4_4_DDR

enumerator kFlashInstMode_DPI_SDR

enumerator kFlashInstMode_DPI_DDR

enumerator kFlashInstMode_QPI_SDR

enumerator kFlashInstMode_QPI_DDR

enumerator kFlashInstMode_OPI_SDR

enumerator kFlashInstMode_OPI_DDR

flash reset sequence of standard SPI mode.

Values:

enumerator kRestoreSeqence_None

enumerator kRestoreSeqence_QPI_4_0xFFs

enumerator kRestoreSeqence_QPI_Mode_0x00

enumerator kRestoreSeqence_Send_F0

enumerator kRestoreSeqence_Send_66_99

enumerator kRestoreSeqence_Send_6699_9966

enumerator kRestoreSeqence_Send_06_FF

enumerator kRestoreSeqence_QPI_5_0xFFs

enumerator kRestoreSeqence_Send_QPI_8_0xFFs

enumerator kRestoreSeqence_Wakep_0xAB

enumerator kRestoreSeqence_Wakep_0xAB_54

typedef enum *_rstctl_reset_source* rstctl_reset_source_t

Enumeration for system reset status bits.

Defines the enumeration for system reset status bits in SYSRSTSTAT register

typedef enum *_RSTCTL_RSTn* RSTCTL_RSTn_t

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in RSTCLTx registers

typedef union *_FLASH_run_context_t* FLASH_run_context_t

Flash context.

Defines the flash context parameter used by ROM.

typedef *RSTCTL_RSTn_t* reset_ip_name_t

IP reset handle.

void RESET_SetPeripheralReset(*reset_ip_name_t* peripheral)

Assert reset to peripheral.

Asserts reset signal to specified peripheral module.

Parameters

- peripheral – Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_ClearPeripheralReset(*reset_ip_name_t* peripheral)

Clear reset to peripheral.

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

- peripheral – Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_PeripheralReset(*reset_ip_name_t* peripheral)

Reset peripheral module.

Reset peripheral module.

Parameters

- peripheral – Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void RESET_ReleasePeripheralReset(*reset_ip_name_t* peripheral)

Release peripheral module.

Release peripheral module.

Parameters

- peripheral – Peripheral to release. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline uint32_t RESET_GetPreviousResetSources(void)

Gets the reset source status which caused a previous reset.

This function gets the status of the latest reset event. Use source masks defined in the rstctl_reset_source_t to get the desired source status.

Returns

All reset source status bit map.

static inline void RESET_ClearResetSources(uint32_t sourceMasks)

Clears the reset source status.

This function clears the system reset flags indicated by source masks.

Parameters

- sourceMasks – reset source status bit map

static inline uint32_t RESET_GetDomainResetStatus(void)

Gets the reset status of domains.

This function gets the reset status of domains.

Returns

All reset source status bit map.

FSL_RESET_DRIVER_VERSION

reset driver version 2.1.0.

RST_CTL0_PSCCTL0

Reset control registers index.

RST_CTL0_PSCCTL1

RST_CTL0_PSCCTL2

RST_CTL0_PSCCTL3

RST_CTL0_PSCCTL4

RST_CTL0_PSCCTL5

RST_CTL1_PSCCTL0

RST_CTL2_PSCCTL0

RST_CTL3_PSCCTL0

RST_CTL3_PSCCTL1

RST_CTL4_PSCCTL0

RST_CTL4_PSCCTL1

ADC_RSTS

Array initializers with peripheral reset bits

SDADC_RSTS

CRC_RSTS

CTIMER_RSTS

MIPI_DSI_RSTS

LCDIF_RSTS

LP_FLEXCOMM_RSTS

FLEXIO_RSTS

FREQME_RSTS_N

XSPI_RSTS

GPIO_RSTS

I3C_RSTS

MRT_RSTS

PINT_RSTS

PNGDEC_RSTS

JPEGDEC_RSTS

SCT_RSTS

SEMA42_RSTS

SAI_RSTS

USDHC_RSTS

UTICK_RSTS

uint8_t reseverd0
: Reserved.

uint8_t current_mode
: Flash device current running mode.

uint8_t reserved1
: Reserved.

uint8_t restore_sequence
: Flash reset sequence of standard SPI mode.

struct *_FLASH_run_context_t* B

uint32_t U

union _FLASH_run_context_t
: *#include <fsl_reset.h>* Flash context.

  Defines the flash context parameter used by ROM.

struct B

Public Members

uint8_t reseverd0
Reserved.

uint8_t current_mode
Flash device current running mode.

uint8_t reserved1
Reserved.

uint8_t restore_sequence
Flash reset sequence of standard SPI mode.

2.73 Runbootloader

typedef struct *_user_app_boot_invoke_option* user_app_boot_invoke_option_t

void bootloader_user_entry(void *arg)
Run the Bootloader API to force into the ISP mode base on the user arg.

Parameters

- arg – Indicates API prototype fields definition. Refer to the above user_app_boot_invoke_option_t structure

uint32_t bootloader_version(void)
Get Bootloader version.

Return values
Bootloader – version.

const char *bootloader_copyright(void)
Get Bootloader Copyright.

Return values
Pointer – to Bootloader copyright header

uint32_t reserved0

uint32_t recovery_boot_cfg0

uint32_t recovery_boot_cfg1

uint32_t reserved1

uint32_t instance

uint32_t boot_interface

uint32_t mode

uint32_t tag

struct *_user_app_boot_invoke_option* B

uint32_t U

union *_user_app_boot_invoke_option* option

struct _user_app_boot_invoke_option
#include <fsl_romapi.h>

union option

Public Members

struct *_user_app_boot_invoke_option* B

uint32_t U

struct B

2.74 SAI: Serial Audio Interface

2.75 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 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

2.76 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

2.77 Sbloader

enum _sectionType
sb3 section definitions

section type

Values:

enumerator kSectionNone
end or invalid

enumerator kSectionDataRange

enumerator kSectionDiffUpdate

enumerator kSectionDDRConfig

enumerator kSectionRegister

Values:

enumerator kFwVerChk_Id_none

enumerator kFwVerChk_Id_nonsecure

enumerator kFwVerChk_Id_secure

enum _loader_command_sb3
loader command enum

Values:

enumerator kSB3_CmdInvalid

enumerator kSB3_CmdErase

enumerator kSB3_CmdLoad

enumerator kSB3_CmdExecute

enumerator kSB3_CmdCall

enumerator kSB3_CmdProgramFuse

enumerator kSB3_CmdProgramIFR

enumerator kSB3_CmdLoadCmac

enumerator kSB3_CmdCopy

enumerator kSB3_CmdLoadHashLocking

enumerator kSB3_CmdLoadKeyBlob

enumerator kSB3_CmdConfigMem

enumerator kSB3_CmdFillMem

enumerator kSB3_CmdFwVerCheck

enum kb_operation_t

Details of the operation to be performed by the ROM.

The kRomAuthenticateImage operation requires the entire signed image to be available to the application.

Values:

enumerator kRomAuthenticateImage

Authenticate a signed image.

enumerator kRomLoadImage

Load SB file.

enumerator kRomOperationCount

typedef uint8_t chunk_v3_t[16]

typedef struct *_ldr_buf* ldr_buf_t

typedef struct *_ldr_Context_v3* ldr_Context_v3_t

Provides forward reference to the loader context definition.

typedef *status_t* (*pLdrFnc_v3_t)(*ldr_Context_v3_t* *content)

Function pointer definition for all loader action functions.

typedef enum *_sectionType* section_type_t

sb3 section definitions

section type

typedef struct *range_header* sb3_data_range_header_t

section data range structure

typedef struct *range_header_expansion* sb3_data_range_expansion_t

typedef struct *copy_memory_expansion* sb3_copy_memory_expansion_t

typedef struct *copy* sb3_copy_memory_t

typedef struct *load_keyblob* sb3_load_keyblob_t

typedef struct *fill_memory_expansion* sb3_fill_memory_expansion_t

typedef struct *fill_memory* sb3_fill_memory_t

typedef struct *config_memory* sb3_config_memory_t

typedef struct *fw_ver_check* sb3_fw_ver_check_t

typedef struct *section_header* sb3_section_header_t

sb3 DATA section header format

typedef enum *_loader_command_sb3* sb3_cmd_t

loader command enum

SB3_BYTES_PER_CHUNK

Defines the number of bytes in a cipher block (chunk). This is dictated by the encryption algorithm.

SB3_DATA_RANGE_HEADER_FLAGS_ERASE_MASK
bit 0

SB3_DATA_RANGE_HEADER_FLAGS_LOAD_MASK
bit 1

SBLOADER_V3_CMD_SET_ALL
The all of the allowed command.

SBLOADER_V3_CMD_SET_IN_ISP_MODE
The allowed command set in ISP mode.

SBLOADER_V3_CMD_SET_IN_REC_MODE
The allowed command set in recovery mode.

SB3_DATA_BUFFER_SIZE_IN_BYTE

struct _ldr_buf
#include <fsl_sbloader_v3.h>

struct range_header
#include <fsl_sbloader_v3.h> section data range structure

struct range_header_expansion
#include <fsl_sbloader_v3.h>

struct copy_memory_expansion
#include <fsl_sbloader_v3.h>

struct copy
#include <fsl_sbloader_v3.h>

struct load_keyblob
#include <fsl_sbloader_v3.h>

struct fill_memory_expansion
#include <fsl_sbloader_v3.h>

Public Members

uint32_t pattern
word to be used as pattern

struct fill_memory
#include <fsl_sbloader_v3.h>

struct config_memory
#include <fsl_sbloader_v3.h>

Public Members

uint32_t address
address of config blob

struct fw_ver_check
#include <fsl_sbloader_v3.h>

struct section_header
#include <fsl_sbloader_v3.h> sb3 DATA section header format

struct kb_region_t
#include <fsl_sbloader_v3.h> Memory region definition.

struct kb_load_sb_t
#include <fsl_sbloader_v3.h>

struct kb_authenticate_t
#include <fsl_sbloader_v3.h>

struct kb_options_t
#include <fsl_sbloader_v3.h>

Public Members

uint32_t version
Should be set to #kKbootApiVersion.

uint8_t *buffer
Caller-provided buffer used by Kboot.

struct _ldr_Context_v3
#include <fsl_sbloader_v3.h> Loader context definition.

Public Members

pLdrFnc_v3_t Action
pointer to loader action function

uint32_t block_size
size of each block in bytes

uint32_t block_data_size
data size in bytes (NBOOT_SB3_CHUNK_SIZE_IN_BYTES)

uint32_t block_data_total
data max size in bytes (block_size * data_size

uint32_t block_buffer_size
block0 and block size

uint32_t processedBlocks
will be used for both block0 and blockx

bool in_data_block
data block offset in a block. in progress of handling a data block within a block

bool in_data_section
in progress of handling a data section within a data block

bool in_data_range
in progress of handling a data range within a data section

uint32_t commandSet
support command set during sb file handling

uint8_t data_buffer[(*MAX*(128, NBOOT_KEY_BLOB_SIZE_IN_BYTE_MAX))]
temporary data buffer

kb_options_t fromAPI
options from ROM API

union ___unnamed92___

Public Members

kb_authenticate_t authenticate

Settings for #kKbootAuthenticate operation.

kb_load_sb_t loadSB

Settings for #kKbootLoadSB operation.

2.78 SCTimer: SCTimer/PWM (SCT)

status_t SCTIMER_Init(SCT_Type *base, const *sctimer_config_t* *config)

Ungates the SCTimer clock and configures the peripheral for basic operation.

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

Parameters

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

Returns

kStatus_Success indicates success; Else indicates failure.

void SCTIMER_Deinit(SCT_Type *base)

Gates the SCTimer clock.

Parameters

- base – SCTimer peripheral base address

void SCTIMER_GetDefaultConfig(*sctimer_config_t* *config)

Fills in the SCTimer configuration structure with the default settings.

The default values are:

```
config->enableCounterUnify = true;
config->clockMode = kSCTIMER_System_ClockMode;
config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
config->enableBidirection_l = false;
config->enableBidirection_h = false;
config->prescale_l = 0U;
config->prescale_h = 0U;
config->outInitState = 0U;
config->inputsync  = 0xFU;
```

Parameters

- config – Pointer to the user configuration structure.

status_t SCTIMER_SetupPwm(SCT_Type *base, const *sctimer_pwm_signal_param_t* *pwmParams, *sctimer_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event)

Configures the PWM signal parameters.

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events

are enabled for the same state. The state number can be retrieved by calling the function SCTIMER_GetCurrentStateNumber(). The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

Note: When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's pwmFreq_Hz.

Parameters

- base – SCTimer peripheral base address
- pwmParams – PWM parameters to configure the output
- mode – PWM operation mode, options available in enumeration sctimer_pwm_mode_t
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – SCTimer counter clock in Hz
- event – Pointer to a variable where the PWM period event number is stored

Returns

kStatus_Success on success kStatus_Fail If we have hit the limit in terms of number of events created or if an incorrect PWM dutycylce is passed in.

void SCTIMER_UpdatePwmDutycycle(SCT_Type *base, *sctimer_out_t* output, uint8_t dutyCyclePercent, uint32_t event)

Updates the duty cycle of an active PWM signal.

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

Parameters

- base – SCTimer peripheral base address
- output – The output to configure
- dutyCyclePercent – New PWM pulse width; the value should be between 1 to 100
- event – Event number associated with this PWM signal. This was returned to the user by the function SCTIMER_SetupPwm().

static inline void SCTIMER_EnableInterrupts(SCT_Type *base, uint32_t mask)

Enables the selected SCTimer interrupts.

Parameters

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

static inline void SCTIMER_DisableInterrupts(SCT_Type *base, uint32_t mask)

Disables the selected SCTimer interrupts.

Parameters

- base – SCTimer peripheral base address

- mask – The interrupts to enable. This is a logical OR of members of the enumeration sctimer_interrupt_enable_t

static inline uint32_t SCTIMER_GetEnabledInterrupts(SCT_Type *base)

Gets the enabled SCTimer interrupts.

Parameters

- base – SCTimer peripheral base address

Returns

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

static inline uint32_t SCTIMER_GetStatusFlags(SCT_Type *base)

Gets the SCTimer status flags.

Parameters

- base – SCTimer peripheral base address

Returns

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

static inline void SCTIMER_ClearStatusFlags(SCT_Type *base, uint32_t mask)

Clears the SCTimer status flags.

Parameters

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

static inline void SCTIMER_StartTimer(SCT_Type *base, uint32_t countertoStart)

Starts the SCTimer counter.

Note: In 16-bit mode, we can enable both Counter_L and Counter_H, In 32-bit mode, we only can select Counter_U.

Parameters

- base – SCTimer peripheral base address
- countertoStart – The SCTimer counters to enable. This is a logical OR of members of the enumeration sctimer_counter_t.

static inline void SCTIMER_StopTimer(SCT_Type *base, uint32_t countertoStop)

Halts the SCTimer counter.

Parameters

- base – SCTimer peripheral base address
- countertoStop – The SCTimer counters to stop. This is a logical OR of members of the enumeration sctimer_counter_t.

status_t SCTIMER_CreateAndScheduleEvent(SCT_Type *base, *sctimer_event_t* howToMonitor, uint32_t matchValue, uint32_t whichIO, *sctimer_counter_t* whichCounter, uint32_t *event)

Create an event that is triggered on a match or IO and schedule in current state.

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The

event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

- base – SCTimer peripheral base address
- howToMonitor – Event type; options are available in the enumeration sctimer_interrupt_enable_t
- matchValue – The match value that will be programmed to a match register
- whichIO – The input or output that will be involved in event triggering. This field is ignored if the event type is "match only"
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Pointer to a variable where the new event number is stored

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

void SCTIMER_ScheduleEvent(SCT_Type *base, uint32_t event)

Enable an event in the current state.

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function SCTIMER_SetupPwm() or function SCTIMER_CreateAndScheduleEvent() .

Parameters

- base – SCTimer peripheral base address
- event – Event number to enable in the current state

status_t SCTIMER_IncreaseState(SCT_Type *base)

Increase the state by 1.

All future events created by calling the function SCTIMER_ScheduleEvent() will be enabled in this new state.

Parameters

- base – SCTimer peripheral base address

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of states used

uint32_t SCTIMER_GetCurrentState(SCT_Type *base)

Provides the current state.

User can use this to set the next state by calling the function SCTIMER_SetupNextStateAction().

Parameters

- base – SCTimer peripheral base address

Returns

The current state

static inline void SCTIMER_SetCounterState(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t state)

Set the counter current state.

The function is to set the state variable bit field of STATE register. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- state – The counter current state number (only support range from 0~31).

static inline uint16_t SCTIMER_GetCounterState(SCT_Type *base, *sctimer_counter_t* whichCounter)

Get the counter current state value.

The function is to get the state variable bit field of STATE register.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The the counter current state value.

status_t SCTIMER_SetupCaptureAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t *captureRegister, uint32_t event)

Setup capture of the counter value on trigger of a selected event.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- captureRegister – Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
- event – Event number that will trigger the capture

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of match/capture registers available

void SCTIMER_SetCallback(SCT_Type *base, *sctimer_event_callback_t* callback, uint32_t event)

Receive noticification when the event trigger an interrupt.

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

- base – SCTimer peripheral base address
- event – Event number that will trigger the interrupt
- callback – Function to invoke when the event is triggered

static inline void SCTIMER_SetupStateLdMethodAction(SCT_Type *base, uint32_t event, bool fgLoad)

Change the load method of transition to the specified state.

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- event – Event number that will change the method to trigger the state transition
- fgLoad – The method to load highest-numbered event occurring for that state to the STATE register.
  - true: Load the STATEV value to STATE when the event occurs to be the next state.
  - false: Add the STATEV value to STATE when the event occurs to be the next state.

static inline void SCTIMER_SetupNextStateActionwithLdMethod(SCT_Type *base, uint32_t nextState, uint32_t event, bool fgLoad)

Transition to the specified state with Load method.

This transition will be triggered by the event number that is passed in by the user, the method decide how to load the highest-numbered event occurring for that state to the STATE register.

Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition
- fgLoad – The method to load the highest-numbered event occurring for that state to the STATE register.
  - true: Load the STATEV value to STATE when the event occurs to be the next state.
  - false: Add the STATEV value to STATE when the event occurs to be the next state.

static inline void SCTIMER_SetupNextStateAction(SCT_Type *base, uint32_t nextState, uint32_t event)

Transition to the specified state.

Deprecated:

Do not use this function. It has been superceded by SCTIMER_SetupNextStateActionwithLdMethod

This transition will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition

static inline void SCTIMER_SetupEventActiveDirection(SCT_Type *base, *sctimer_event_active_direction_t* activeDirection, uint32_t event)

Setup event active direction when the counters are operating in BIDIR mode.

Parameters

- base – SCTimer peripheral base address
- activeDirection – Event generation active direction, see sctimer_event_active_direction_t.
- event – Event number that need setup the active direction.

static inline void SCTIMER_SetupOutputSetAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Set the Output.

This output will be set when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to set
- event – Event number that will trigger the output change

static inline void SCTIMER_SetupOutputClearAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Clear the Output.

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to clear
- event – Event number that will trigger the output change

void SCTIMER_SetupOutputToggleAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Toggle the output level.

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to toggle
- event – Event number that will trigger the output change

static inline void SCTIMER_SetupCounterLimitAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Limit the running counter.

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be limited

static inline void SCTIMER_SetupCounterStopAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Stop the running counter.

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be stopped

static inline void SCTIMER_SetupCounterStartAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Re-start the stopped counter.

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to re-start

static inline void SCTIMER_SetupCounterHaltAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Halt the running counter.

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the SCTIMER_StartTimer() function.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be halted

static inline void SCTIMER_SetupDmaTriggerAction(SCT_Type *base, uint32_t dmaNumber, uint32_t event)

Generate a DMA request.

DMA request will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- dmaNumber – The DMA request to generate
- event – Event number that will trigger the DMA request

static inline void SCTIMER_SetCOUNTValue(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t value)

Set the value of counter.

The function is to set the value of Count register, Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- value – the counter value update to the COUNT register.

static inline uint32_t SCTIMER_GetCOUNTValue(SCT_Type *base, *sctimer_counter_t* whichCounter)

Get the value of counter.

The function is to read the value of Count register, software can read the counter registers at any time..

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The value of counter selected.

static inline void SCTIMER_SetEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Set the state mask bit field of EV_STATE register.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be set.
- state – The state value in which the event is enabled to occur.

static inline void SCTIMER_ClearEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Clear the state mask bit field of EV_STATE register.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be clear.
- state – The state value in which the event is disabled to occur.

static inline bool SCTIMER_GetEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Get the state mask bit field of EV_STATE register.

Note: This function is to check whether the event is enabled in a specific state.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be read.
- state – The state value.

Returns

The the state mask bit field of EV_STATE register.

- true: The event is enable in state.
- false: The event is disable in state.

static inline uint32_t SCTIMER_GetCaptureValue(SCT_Type *base, *sctimer_counter_t* whichCounter, uint8_t capChannel)

Get the value of capture register.

This function returns the captured value upon occurrence of the events selected by the corresponding Capture Control registers occurred.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- capChannel – SCTimer capture register of capture channel.

Returns

The SCTimer counter value at which this register was last captured.

void SCTIMER_EventHandleIRQ(SCT_Type *base)

SCTimer interrupt handler.

Parameters

- base – SCTimer peripheral base address.

FSL_SCTIMER_DRIVER_VERSION

Version

enum _sctimer_pwm_mode

SCTimer PWM operation modes.

Values:

enumerator kSCTIMER_EdgeAlignedPwm

Edge-aligned PWM

enumerator kSCTIMER_CenterAlignedPwm

Center-aligned PWM

enum _sctimer_counter

SCTimer counters type.

Values:

enumerator kSCTIMER_Counter_L

16-bit Low counter.

enumerator kSCTIMER_Counter_H

16-bit High counter.

enumerator kSCTIMER_Counter_U

32-bit Unified counter.

enum _sctimer_input

List of SCTimer input pins.

Values:

enumerator kSCTIMER_Input_0

SCTIMER input 0

enumerator kSCTIMER_Input_1

SCTIMER input 1

enumerator kSCTIMER_Input_2
SCTIMER input 2

enumerator kSCTIMER_Input_3
SCTIMER input 3

enumerator kSCTIMER_Input_4
SCTIMER input 4

enumerator kSCTIMER_Input_5
SCTIMER input 5

enumerator kSCTIMER_Input_6
SCTIMER input 6

enumerator kSCTIMER_Input_7
SCTIMER input 7

enum _sctimer_out
List of SCTimer output pins.

Values:

enumerator kSCTIMER_Out_0
SCTIMER output 0

enumerator kSCTIMER_Out_1
SCTIMER output 1

enumerator kSCTIMER_Out_2
SCTIMER output 2

enumerator kSCTIMER_Out_3
SCTIMER output 3

enumerator kSCTIMER_Out_4
SCTIMER output 4

enumerator kSCTIMER_Out_5
SCTIMER output 5

enumerator kSCTIMER_Out_6
SCTIMER output 6

enumerator kSCTIMER_Out_7
SCTIMER output 7

enumerator kSCTIMER_Out_8
SCTIMER output 8

enumerator kSCTIMER_Out_9
SCTIMER output 9

enum _sctimer_pwm_level_select
SCTimer PWM output pulse mode: high-true, low-true or no output.

Values:

enumerator kSCTIMER_LowTrue
Low true pulses

enumerator kSCTIMER_HighTrue
High true pulses

enum _sctimer_clock_mode

SCTimer clock mode options.

Values:

enumerator kSCTIMER_System_ClockMode

System Clock Mode

enumerator kSCTIMER_Sampled_ClockMode

Sampled System Clock Mode

enumerator kSCTIMER_Input_ClockMode

SCT Input Clock Mode

enumerator kSCTIMER_Asynchronous_ClockMode

Asynchronous Mode

enum _sctimer_clock_select

SCTimer clock select options.

Values:

enumerator kSCTIMER_Clock_On_Rise_Input_0

Rising edges on input 0

enumerator kSCTIMER_Clock_On_Fall_Input_0

Falling edges on input 0

enumerator kSCTIMER_Clock_On_Rise_Input_1

Rising edges on input 1

enumerator kSCTIMER_Clock_On_Fall_Input_1

Falling edges on input 1

enumerator kSCTIMER_Clock_On_Rise_Input_2

Rising edges on input 2

enumerator kSCTIMER_Clock_On_Fall_Input_2

Falling edges on input 2

enumerator kSCTIMER_Clock_On_Rise_Input_3

Rising edges on input 3

enumerator kSCTIMER_Clock_On_Fall_Input_3

Falling edges on input 3

enumerator kSCTIMER_Clock_On_Rise_Input_4

Rising edges on input 4

enumerator kSCTIMER_Clock_On_Fall_Input_4

Falling edges on input 4

enumerator kSCTIMER_Clock_On_Rise_Input_5

Rising edges on input 5

enumerator kSCTIMER_Clock_On_Fall_Input_5

Falling edges on input 5

enumerator kSCTIMER_Clock_On_Rise_Input_6

Rising edges on input 6

enumerator kSCTIMER_Clock_On_Fall_Input_6

Falling edges on input 6

enumerator kSCTIMER_Clock_On_Rise_Input_7
Rising edges on input 7

enumerator kSCTIMER_Clock_On_Fall_Input_7
Falling edges on input 7

enum _sctimer_conflict_resolution
SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Values:

enumerator kSCTIMER_ResolveNone
No change

enumerator kSCTIMER_ResolveSet
Set output

enumerator kSCTIMER_ResolveClear
Clear output

enumerator kSCTIMER_ResolveToggle
Toggle output

enum _sctimer_event_active_direction
List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

Values:

enumerator kSCTIMER_ActiveIndependent
This event is triggered regardless of the count direction.

enumerator kSCTIMER_ActiveInCountUp
This event is triggered only during up-counting when BIDIR = 1.

enumerator kSCTIMER_ActiveInCountDown
This event is triggered only during down-counting when BIDIR = 1.

enum _sctimer_event
List of SCTimer event types.

Values:

enumerator kSCTIMER_InputLowOrMatchEvent

enumerator kSCTIMER_InputRiseOrMatchEvent

enumerator kSCTIMER_InputFallOrMatchEvent

enumerator kSCTIMER_InputHighOrMatchEvent

enumerator kSCTIMER_MatchEventOnly

enumerator kSCTIMER_InputLowEvent

enumerator kSCTIMER_InputRiseEvent

enumerator kSCTIMER_InputFallEvent

enumerator kSCTIMER_InputHighEvent

enumerator kSCTIMER_InputLowAndMatchEvent

enumerator kSCTIMER_InputRiseAndMatchEvent

enumerator kSCTIMER_InputFallAndMatchEvent

enumerator kSCTIMER_InputHighAndMatchEvent

enumerator kSCTIMER_OutputLowOrMatchEvent

enumerator kSCTIMER_OutputRiseOrMatchEvent

enumerator kSCTIMER_OutputFallOrMatchEvent

enumerator kSCTIMER_OutputHighOrMatchEvent

enumerator kSCTIMER_OutputLowEvent

enumerator kSCTIMER_OutputRiseEvent

enumerator kSCTIMER_OutputFallEvent

enumerator kSCTIMER_OutputHighEvent

enumerator kSCTIMER_OutputLowAndMatchEvent

enumerator kSCTIMER_OutputRiseAndMatchEvent

enumerator kSCTIMER_OutputFallAndMatchEvent

enumerator kSCTIMER_OutputHighAndMatchEvent

enum _sctimer_interrupt_enable

List of SCTimer interrupts.

Values:

enumerator kSCTIMER_Event0InterruptEnable

Event 0 interrupt

enumerator kSCTIMER_Event1InterruptEnable

Event 1 interrupt

enumerator kSCTIMER_Event2InterruptEnable

Event 2 interrupt

enumerator kSCTIMER_Event3InterruptEnable

Event 3 interrupt

enumerator kSCTIMER_Event4InterruptEnable

Event 4 interrupt

enumerator kSCTIMER_Event5InterruptEnable

Event 5 interrupt

enumerator kSCTIMER_Event6InterruptEnable

Event 6 interrupt

enumerator kSCTIMER_Event7InterruptEnable

Event 7 interrupt

enumerator kSCTIMER_Event8InterruptEnable

Event 8 interrupt

enumerator kSCTIMER_Event9InterruptEnable

Event 9 interrupt

enumerator kSCTIMER_Event10InterruptEnable
Event 10 interrupt

enumerator kSCTIMER_Event11InterruptEnable
Event 11 interrupt

enumerator kSCTIMER_Event12InterruptEnable
Event 12 interrupt

enum _sctimer_status_flags
List of SCTimer flags.

Values:

enumerator kSCTIMER_Event0Flag
Event 0 Flag

enumerator kSCTIMER_Event1Flag
Event 1 Flag

enumerator kSCTIMER_Event2Flag
Event 2 Flag

enumerator kSCTIMER_Event3Flag
Event 3 Flag

enumerator kSCTIMER_Event4Flag
Event 4 Flag

enumerator kSCTIMER_Event5Flag
Event 5 Flag

enumerator kSCTIMER_Event6Flag
Event 6 Flag

enumerator kSCTIMER_Event7Flag
Event 7 Flag

enumerator kSCTIMER_Event8Flag
Event 8 Flag

enumerator kSCTIMER_Event9Flag
Event 9 Flag

enumerator kSCTIMER_Event10Flag
Event 10 Flag

enumerator kSCTIMER_Event11Flag
Event 11 Flag

enumerator kSCTIMER_Event12Flag
Event 12 Flag

enumerator kSCTIMER_BusErrorLFlag
Bus error due to write when L counter was not halted

enumerator kSCTIMER_BusErrorHFlag
Bus error due to write when H counter was not halted

typedef enum *_sctimer_pwm_mode* sctimer_pwm_mode_t
SCTimer PWM operation modes.

typedef enum *_sctimer_counter* sctimer_counter_t
SCTimer counters type.

typedef enum *_sctimer_input* sctimer_input_t
List of SCTimer input pins.

typedef enum *_sctimer_out* sctimer_out_t
List of SCTimer output pins.

typedef enum *_sctimer_pwm_level_select* sctimer_pwm_level_select_t
SCTimer PWM output pulse mode: high-true, low-true or no output.

typedef struct *_sctimer_pwm_signal_param* sctimer_pwm_signal_param_t
Options to configure a SCTimer PWM signal.

typedef enum *_sctimer_clock_mode* sctimer_clock_mode_t
SCTimer clock mode options.

typedef enum *_sctimer_clock_select* sctimer_clock_select_t
SCTimer clock select options.

typedef enum *_sctimer_conflict_resolution* sctimer_conflict_resolution_t
SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

typedef enum *_sctimer_event_active_direction* sctimer_event_active_direction_t
List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

typedef enum *_sctimer_event* sctimer_event_t
List of SCTimer event types.

typedef void (*sctimer_event_callback_t)(void)
SCTimer callback typedef.

typedef enum *_sctimer_interrupt_enable* sctimer_interrupt_enable_t
List of SCTimer interrupts.

typedef enum *_sctimer_status_flags* sctimer_status_flags_t
List of SCTimer flags.

typedef struct *_sctimer_config* sctimer_config_t
SCTimer configuration structure.

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

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

SCT_EV_STATE_STATEMSKn(x)

struct _sctimer_pwm_signal_param
#include <fsl_sctimer.h> Options to configure a SCTimer PWM signal.

Public Members

sctimer_out_t output
The output pin to use to generate the PWM signal

sctimer_pwm_level_select_t level

PWM output active level select.

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).

struct __sctimer__config

#include <fsl_sctimer.h> SCTimer configuration structure.

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

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

Public Members

bool enableCounterUnify

true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters. User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

sctimer_clock_mode_t clockMode

SCT clock mode value

sctimer_clock_select_t clockSelect

SCT clock select value

bool enableBidirection_l

true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter

bool enableBidirection_h

true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter. This field is used only if the enableCounterUnify is set to false

uint8_t prescale_l

Prescale value to produce the L or unified counter clock

uint8_t prescale_h

Prescale value to produce the H counter clock. This field is used only if the enableCounterUnify is set to false

uint8_t outInitState

Defines the initial output value

uint8_t inputsync

SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16. it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ~bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNC0 (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 « INPUTSYNC2) | (1 « INPUTSYNC3);

2.79 SDADC: SDADC Module

void SDADC_GetDefaultConfig(*sdadc_config_t* *config)

This function is used to get available predefined configurations for the SDADC initialization.

Parameters

- config – Pointer to the SDADC configuration structure, please refer to sdadc_config_t for details.

void SDADC_Init(SDADC_Type *base, const *sdadc_config_t* *config)

This function is used to initialize the SDADC.

Parameters

- base – SDADC peripheral base address.
- config – Pointer to the SDADC configuration structure, please refer to sdadc_config_t for details.

void SDADC_Deinit(SDADC_Type *base, const *sdadc_config_t* *config)

This function is used to de-initialize the SDADC.

Parameters

- base – SDADC peripheral base address.
- config – Pointer to the SDADC configuration structure, please refer to sdadc_config_t for details.

void SDADC_DoInitPowerUp(SDADC_Type *base, *sdadc_config_t* *config, uint32_t clock)

This function is used to power up the SDADC in the initialization state.

Parameters

- base – SDADC peripheral base address.
- config – Pointer to the SDADC configuration structure, please refer to sdadc_config_t for details.
- clock – Core clock frequency with Hz.

static inline void SDADC_ControlModulatorPowerEnable(SDADC_Type *base,
sdadc_channel_number_t number,
sdadc_channel_type_t type, bool enable)

This function is used to enable or disable SDADC specific channel's P-Side or/and N-Side sigma-delta modulator power.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlReferencePowerEnable(SDADC_Type *base, bool enable)

This function is used to enable or disable the reference power.

Parameters

- base – SDADC peripheral base address.
- enable – Indicates enable or disable the reference power.
  - **true** Enable

- **false** Disable

static inline void SDADC_ConfigureReferenceMode(SDADC_Type *base, *sdadc_ref_mode_t* referenceMode)

This function is used to configure the reference mode.

Note: Setting the reference to fast charge mode is mainly used during startup where fast startup is required, the current consumption in this mode is the largest, so, it is recommended to switch to other modes to minimize current consumption. If the ADC works on the single-end mode, recommend switching to the low-noise mode, if the ADC works on the differential mode, recommend switching to the retain mode.

Parameters

- base – SDADC peripheral base address.
- referenceMode – Reference mode to be configured, see sdadc_ref_mode_t for details.

static inline void SDADC_ControlLdoPowerEnable(SDADC_Type *base, bool enable)

This function is used to enable or disable the LDO power.

Parameters

- base – SDADC peripheral base address.
- enable – Indicates enable or disable the LDO power.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ConfigureLdoDriveCapability(SDADC_Type *base, *sdadc_ldo_drive_capability_t* capability)

This function is used to configure the drive capability.

Parameters

- base – SDADC peripheral base address.
- capability – SDADC LDO drive capability to be set, see sdadc_ldo_drive_capability_t for details.

static inline void SDADC_ConfigureChannelMode(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_mode_t* mode)

This function is used to configure the channel mode.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- mode – channel mode, refer sdadc_channel_mode_t for details.

static inline void SDADC_ControlDacCompensationEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the DAC compensation function.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.

- type – channel type, refer sdadc_channel_type_t for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlModulatorDcLoopEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the modulator DC Loop function.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlDecimatorDcFilterEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the decimator DC Filter function.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlDecimatorOutputInvertEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the decimator output invert.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ConfigureDecimatorVolume(SDADC_Type *base,
sdadc_channel_number_t number,
sdadc_channel_type_t type,
sdadc_volume_control_t volume)

This function is used to configure the decimator volume.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- volume – Volume level, see sdadc_volume_control_t for details.

static inline void SDADC_ConfigureDecimatorSampleRate(SDADC_Type *base,
sdadc_channel_number_t number,
sdadc_channel_type_t type,
sdadc_sample_rate_t sampleRate)

This function is used to configure the decimator sample rate.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- sampleRate – Sample rate, see sdadc_sample_rate_t for details.

static inline void SDADC_ConfigureDitherAmplitude(SDADC_Type *base,
sdadc_channel_number_t number, uint8_t ditherAmplitude)

This function is used to set the dither amplitude.

In the ADC conversion process, adding dither can improve the ADC dynamic performance, but too large a dither will cause the input signal to overflow, the user needs to set the appropriate dither amplitude according to their own needs.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- ditherAmplitude – The dither amplitude that will be introduced, the range is from 000b to 111b, 000b means disable the dither.

static inline void SDADC_ControlModulatorReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool reset)

This function is used to reset or release reset for the SDADC specific channel's P-Side or N-Side sigma-delta modulator.

Note: Needs to be reset for at least 15us after start-up, and then release the reset.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- reset – Indicates reset or release reset for the sigma-delta modulator.

- **true** Reset
- **false** Release reset

static inline void SDADC_ControlDecimatorReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool reset)

This function is used to reset or release reset for the SDADC channeln's P-Side or/and N-Side decimator.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- reset – Indicates reset or release reset for the decimator.
  - **true** Reset
  - **false** Release reset

static inline void SDADC_ConfigureModulatorDcLoopReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool reset)

This function is used to reset or release reset for the SDADC channeln's P-Side or/and N-Side madulator DC Loop.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- reset – Indicates whether reset or release reset for the madulator DC Loop.
  - **true** Reset
  - **false** Release reset

static inline void SDADC_ControlGlobalFifoIntEnable(SDADC_Type *base, bool enable)

This function is used to enable or disable the global FIFO full interrupt.

Parameters

- base – SDADC peripheral base address.
- enable – Indicates whether enable or disable the global FIFO full interrupt.
  - **true** Enable interrupt
  - **false** Disable interrupt

static inline bool SDADC_CheckGlobalFifoInterrupted(SDADC_Type *base)

This function is used to check whether the FIFO full interrupt has occurred.

Parameters

- base – SDADC peripheral base address.

Returns

The FIFO full interrupt status flag.

- **true** The FIFO full interrupt has occurred.
- **false** The FIFO full interrupt has not occurred.

static inline void SDADC_ClearGlobalFifoIntStatusFlag(SDADC_Type *base)

This function is used to clear the FIFO full interrupt status flag.

Parameters

- base – SDADC peripheral base address.

static inline void SDADC_ControlFifoIntEnable(SDADC_Type *base, uint32_t mask, bool enable)

This function is used to enable or disable the FIFO full interrupt.

Parameters

- base – SDADC peripheral base address.
- mask – Mask value to enable or disable interrupt, please refer to _sdadc_fifo_int_enable for details.
- enable – Indicates enable or disable the FIFO full interrupt.
  - **true** Enable interrupt
  - **false** Disable interrupt

static inline void SDADC_ControlFifoWatermarkEnable(SDADC_Type *base, uint32_t mask, bool enable)

This function is used to enable or disable the FIFO watermark.

Parameters

- base – SDADC peripheral base address.
- mask – Mask value to enable or disable watermark, please refer to _sdadc_fifo_watermark_enable for details.
- enable – Indicates enable or disable the FIFO watermark.
  - **true** Enable watermark
  - **false** Disable watermark

static inline uint8_t SDADC_GetFifoIntStatusFlags(SDADC_Type *base)

This function is used to get the FIFO full interrupt status flags.

Parameters

- base – SDADC peripheral base address.

Returns

Status flags of the FIFO full interrupt.

static inline uint8_t SDADC_GetFifoWatermarkStatusFlags(SDADC_Type *base)

This function is used to get the FIFO watermark status flags.

Parameters

- base – SDADC peripheral base address.

Returns

Status flags of the FIFO watermark.

static inline void SDADC_ConfigureFifoWatermark(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, *sdadc_watermark_t* waterMark)

This function is used to configure FIFO watermark.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.

- type – channel type, refer sdadc_channel_type_t for details.
- waterMark – WaterMark value, see sdadc_watermark_t for details.

static inline void SDADC_ControlFifoReadWriteReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, *sdadc_fifo_reset_operation_t* operation)

This function is used to reset the specific channel side FIFO's read or/and write pointer.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.
- operation – Indicates which reset operation to do, refer sdadc_fifo_reset_operation_t for details.

static inline uint8_t SDADC_GetConvChannelFifoAvailableEntries(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type)

This function is used to get the specific channel side FIFO's available entries.

Note: In the differential mode, the bit stream is converted to 24bit data (FIFO_x_P[23:0]) by the P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.

Returns

The available entries of the specified FIFO, note that only the lower 5 bits of the returned data are valid.

static inline uint32_t SDADC_GetConvChannelFifoRawData(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type)

This function is used to get the specific channel side FIFO's raw data.

Note: In the differential mode, the bit stream is converted to 24bit data (FIFO_x_P[23:0]) by the P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid. So the type that needs to be passed in in differential mode is kSDADC_Channel_PSide_Type.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer sdadc_channel_number_t for details.
- type – channel type, refer sdadc_channel_type_t for details.

Returns

The raw data of the specified FIFO.

void SDADC_CopyConvChannelFifoToBuffer(SDADC_Type *base, *sdadc_channel_group* *group, uint8_t count, void *buffer)

This function is used to copy specific conversion channels' data from FIFO to buffer.

Note: This function will copy all 16 entries in the specified FIFO.

Parameters

- base – SDADC peripheral base address.
- group – Indicates which group of channels data to obtain.
- count – Indicates how many channels in the specified group.
- buffer – The buffer which stores conversion data.

FSL_SDADC_DRIVER_VERSION

SDADC driver version 2.0.1.

enum _sdadc_fifo_int_enable

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO full interrupt enable/disable.

Values:

enumerator kSDADC_Channel0_PSide_FifoFullIntEnable

enumerator kSDADC_Channel0_NSide_FifoFullIntEnable

enumerator kSDADC_Channel1_PSide_FifoFullIntEnable

enumerator kSDADC_Channel1_NSide_FifoFullIntEnable

enumerator kSDADC_Channel2_PSide_FifoFullIntEnable

enumerator kSDADC_Channel2_NSide_FifoFullIntEnable

enumerator kSDADC_Channel3_PSide_FifoFullIntEnable

enumerator kSDADC_Channel3_NSide_FifoFullIntEnable

enum _sdadc_fifo_watermark_enable

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO watermark enable/disable.

Values:

enumerator kSDADC_Channel0_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel0_NSide_FifoWatermarkEnable

enumerator kSDADC_Channel1_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel1_NSide_FifoWatermarkEnable

enumerator kSDADC_Channel2_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel2_NSide_FifoWatermarkEnable

enumerator kSDADC_Channel3_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel3_NSide_FifoWatermarkEnable

enum _sdadc_fifo_int_status_flag

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO full interrupt status flag.

Values:

enumerator kSDADC_Channel0_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel0_NSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel1_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel1_NSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel2_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel2_NSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel3_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel3_NSide_FifoFullIntStatusFlag

enum _sdadc_fifo_watermark_status_flag

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO watermark status flag.

Values:

enumerator kSDADC_Channel0_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel0_NSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel1_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel1_NSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel2_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel2_NSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel3_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel3_NSide_FifoWatermarkStatusFlag

enum _sdadc_channel_mode

This enumeration provides the selection of the SDADC channel mode.

Values:

enumerator kSDADC_SingleEnd_Mode

Single-end mode.

enumerator kSDADC_Differential_Mode

Differential mode.

enum _sdadc_channel_number

This enumeration provides the selection of the SDADC channel number.

Values:

enumerator kSDADC_Channel0

SDADC channel 0.

enumerator kSDADC_Channel1

SDADC channel 1.

enumerator kSDADC_Channel2
SDADC channel 2.

enumerator kSDADC_Channel3
SDADC channel 3.

enum _sdadc_channel_type
This enumeration provides the selection of the SDADC channel type.

Values:

enumerator kSDADC_Channel_NSide_Type
N-side selected.

enumerator kSDADC_Channel_PSide_Type
P-side selected.

enumerator kSDADC_Channel_NandPSide_Type
N- and P-side selected.

enum _sdadc_decimator_sample_rate
This enumeration provides the selection of the SDADC decimator sample rate.

Values:

enumerator kSDADC_DecimatorSampleRate48kHz
48kHz decimator sample rate.

enumerator kSDADC_DecimatorSampleRate32kHz
32kHz decimator sample rate.

enumerator kSDADC_DecimatorSampleRate16kHz
16kHz decimator sample rate.

enum _sdadc_ldo_drive_capability
This enumeration provides the selection of the SDADC LDO drive capability.

Values:

enumerator kSDADC_LdoLowPower
To drive 1 single-end mode or 1 differential mode SDADC.

enumerator kSDADC_LdoHighPower
To drive more than 1 SDADC

enum _sdadc_ref_mode
This enumeration provides the selection of the SDADC reference mode.

The SDADC reference module has four modes, these modes are controlled via REF_CTL register 'fast_ref_enable' and 'pwr_fast_ref_enable' bitfields. 1.Retain mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 0b. 2.Low noise mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 1b. 3.Low power mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 0b. 4.Fast charge mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 1b.

Values:

enumerator kSDADC_RefRetainMode
Reference in retain mode.

enumerator kSDADC_RefLowNoiseMode
Reference in low noise mode.

enumerator kSDADC_RefLowPowerMode
Reference in low power mode.

enumerator kSDADC_RefFastChargeMode
Reference in fast charge mode.

enum _sdadc_fifo_reset_operation
This enumeration provides the selection of the SDADC FIFO write or/and read operation.

Values:

enumerator kSDADC_FifoWriteReset
FIFO write reset.

enumerator kSDADC_FifoReadReset
FIFO read reset.

enumerator kSDADC_FifoWriteAndReadReset
FIFO write and read reset.

typedef enum *_sdadc_channel_mode* sdadc_channel_mode_t
This enumeration provides the selection of the SDADC channel mode.

typedef enum *_sdadc_channel_number* sdadc_channel_number_t
This enumeration provides the selection of the SDADC channel number.

typedef enum *_sdadc_channel_type* sdadc_channel_type_t
This enumeration provides the selection of the SDADC channel type.

typedef enum *_sdadc_decimator_sample_rate* sdadc_decimator_sample_rate_t
This enumeration provides the selection of the SDADC decimator sample rate.

typedef enum *_sdadc_ldo_drive_capability* sdadc_ldo_drive_capability_t
This enumeration provides the selection of the SDADC LDO drive capability.

typedef enum *_sdadc_ref_mode* sdadc_ref_mode_t
This enumeration provides the selection of the SDADC reference mode.

The SDADC reference module has four modes, these modes are controlled via REF_CTL register 'fast_ref_enable' and 'pwr_fast_ref_enable' bitfields. 1.Retain mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 0b. 2.Low noise mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 1b. 3.Low power mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 0b. 4.Fast charge mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 1b.

typedef enum *_sdadc_fifo_reset_operation* sdadc_fifo_reset_operation_t
This enumeration provides the selection of the SDADC FIFO write or/and read operation.

typedef struct *_sdadc_volume* sdadc_volume_control_t
This structure is used to configure the decimator volume (gain).

Note: The gain setting is not applicable for the channel N-Side when executing the differential mode. For the specific gain setting value please refer to the reference manual.

---

typedef struct *_sdadc_sample_rate* sdadc_sample_rate_t
This structure is used to configure the decimator sample rate.

typedef struct *_sdadc_watermark* sdadc_watermark_t
This structure is used to configure the FIFO watermark.

Note: In the differential mode, the bitstream is converted to 24bit data (FIFO_x_P[23:0]) by P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid.

---

typedef struct *_sdadc_channel_config* sdadc_channel_config_t
: This structure is used to configure the SDADC channels.

typedef struct *_sdadc_config* sdadc_config_t
: This structure is used to configure the SDADC module.

typedef struct *_sdadc_channel_group* sdadc_channel_group
: This structure is used to construct channel groups.

SDADC_DECIMATOR_VOL_CTRL_P_MASK(offset)
: Macros are used to configure decimator volume.

SDADC_DECIMATOR_VOL_CTRL_P_SHIFT(offset)

SDADC_DECIMATOR_VOL_CTRL_P(volume, offset)

SDADC_DECIMATOR_VOL_CTRL_N_MASK(offset)

SDADC_DECIMATOR_VOL_CTRL_N_SHIFT(offset)

SDADC_DECIMATOR_VOL_CTRL_N(volume, offset)

SDADC_DECIMATOR_SET_SPEED_P_MASK(offset)
: Macros are used to configure decimator sample rate.

SDADC_DECIMATOR_SET_SPEED_P_SHIFT(offset)

SDADC_DECIMATOR_SET_SPEED_P(rate, offset)

SDADC_DECIMATOR_SET_SPEED_N_MASK(offset)

SDADC_DECIMATOR_SET_SPEED_N_SHIFT(offset)

SDADC_DECIMATOR_SET_SPEED_N(rate, offset)

SDADC_ADC_CTL_2_SET_DITHER_MASK(offset)
: Macros are used to configure the dither amplitude.

SDADC_ADC_CTL_2_SET_DITHER_SHIFT(offset)

SDADC_ADC_CTL_2_SET_DITHER(val, offset)

SDADC_FIFO_WATERMARK_CTL_P_MASK(offset)
: Macros are used to configure FIFO watermark.

SDADC_FIFO_WATERMARK_CTL_P_SHIFT(offset)

SDADC_FIFO_WATERMARK_CTL_P(watermark, offset)

SDADC_FIFO_WATERMARK_CTL_N_MASK(offset)

SDADC_FIFO_WATERMARK_CTL_N_SHIFT(offset)

SDADC_FIFO_WATERMARK_CTL_N(watermark, offset)

SDADC_FIFO_WATERMARK_STATUS_MASK
: Macros are used for FIFO operations.

SDADC_FIFO_WATERMARK_STATUS_SHIFT

SDADC_FIFO_FULL_INT_STATUS_MASK

SDADC_FIFO_FULL_INT_STATUS_SHIFT

SDADC_FIFO_MASK

SDADC_FIFO_SHIFT

SDADC_FIFO_ENTRIES_AVAIL_MASK

SDADC_FIFO_DEPTH

SDADC_FIFO_WIDTH

struct _sdadc_volume

#include <fsl_sdadc.h> This structure is used to configure the decimator volume (gain).

Note: The gain setting is not applicable for the channel N-Side when executing the differential mode. For the specific gain setting value please refer to the reference manual.

Public Members

uint8_t pSideVolume

Sets the gain for the P-Side decimator.

uint8_t nSideVolume

Sets the gain for the N-Side decimator.

struct _sdadc_sample_rate

#include <fsl_sdadc.h> This structure is used to configure the decimator sample rate.

Public Members

sdadc_decimator_sample_rate_t pSideSampleRate

Sets the P-Side decimator sample rate.

sdadc_decimator_sample_rate_t nSideSampleRate

Sets the N-Side decimator sample rate.

struct _sdadc_watermark

#include <fsl_sdadc.h> This structure is used to configure the FIFO watermark.

Note: In the differential mode, the bitstream is converted to 24bit data (FIFO_x_P[23:0]) by P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid.

Public Members

uint8_t pSideWatermark

Sets the channel N-Side FIFO watermark.

uint8_t nSideWatermark

Sets the channel P-Side FIFO watermark.

struct _sdadc_channel_config

#include <fsl_sdadc.h> This structure is used to configure the SDADC channels.

Public Members

sdadc_channel_mode_t mode
Sets channel mode.

sdadc_channel_number_t number
Sets channel number.

sdadc_channel_type_t type
Sets channel type.

sdadc_volume_control_t volume
Sets N- and P-side decimator volume.

sdadc_sample_rate_t samplerate
Sets N- and P-side decimator samplerate.

sdadc_watermark_t watermark
Sets N- and P-side FIFO watermark.

bool enableDacCompensation
Decides whether to enable the compensation feature for the ADCp[3:0] or/and ADCn[3:0] in the single-end mode, is used to avoid large data-dependent ripple on LDO output which will impact the ADC performance.

bool enableDCLoop
In some cases of the single-end mode, the microphone output is connected directly to ADC input without the coupling capacitor, DC Loop is activated to regulate input common-mode voltage of the ADCp[3:0] or/and ADCn[3:0]. In the single-end mode, there is the possibility to connect MEMS or JFET microphone directly to the input (DC-coupled). In this configuration, DC Loop mode needs to be activated to establish the correct common mode input voltage level at the input pin. The DC Loop function should disabled in the differential mode.

bool enableDcFilter
Decides whether to enable the decimator DC Filter to remove DC components.

bool enablePolarityInvert
Decides whether to enable the polarity invert function, for example: if the input to ADC is a pure sinus signal, then if the polarity invert is enabled, the decimation output will be a sinus signal 180 out of phase with the input.

struct _sdadc_config
#include <fsl_sdadc.h> This structure is used to configure the SDADC module.

Public Members

uint8_t channelCount
Indicates how many channels to execute the conversion.

sdadc_channel_config_t *channelConfig
SDADC channels configuration.

struct _sdadc_channel_group
#include <fsl_sdadc.h> This structure is used to construct channel groups.

Public Members

sdadc_channel_number_t number
Channel number.

sdadc_channel_type_t type
Channel type.

2.80 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.

2.81 Soc_mipi_dsi

FSL_SOC_MIPI_DSI_DRIVER_VERSION

Driver version.

DSI_DPHY_PLL_VCO_MAX

DSI_DPHY_PLL_VCO_MIN

FSL_COMPONENT_ID

2.82 SYSPM: System Performance Monitor

enum _syspm_monitor

syspm select control monitor

Values:

enumerator kSYSPM_Monitor0

Monitor 0

enum _syspm_event

syspm select event

Values:

enumerator kSYSPM_Event1

Event 1

enumerator kSYSPM_Event2

Event 2

enumerator kSYSPM_Event3
Event 3

enum _syspm_mode
syspm set count mode

Values:

enumerator kSYSPM_BothMode
count in both modes

enumerator kSYSPM_UserMode
count only in user mode

enumerator kSYSPM_PrivilegedMode
count only in privileged mode

enum _syspm_startstop_control
syspm start/stop control

Values:

enumerator kSYSPM_Idle
idle >

enumerator kSYSPM_LocalStop
local stop

enumerator kSYSPM_LocalStart
local start

enumerator KSYSPM_EnableTraceControl
enable global TSTART/TSTOP

enumerator kSYSPM_GlobalStart
global stop

enumerator kSYSPM_GlobalStop
global start

typedef enum *_syspm_monitor* syspm_monitor_t
syspm select control monitor

typedef enum *_syspm_event* syspm_event_t
syspm select event

typedef enum *_syspm_mode* syspm_mode_t
syspm set count mode

typedef enum *_syspm_startstop_control* syspm_startstop_control_t
syspm start/stop control

void SYSPM_Init(SYSPM_Type *base)
Initializes the SYSPM.

This function enables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

void SYSPM_Deinit(SYSPM_Type *base)
Deinitializes the SYSPM.

This function disables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

void SYSPM_SelectEvent(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event, uint8_t eventCode)

Select event counters.

Parameters

- base – SYSPM peripheral base address.
- event – syspm select event, see to syspm_event_t.
- eventCode – select which event to be counted in PMECTRx., see to table Events.

void SYSPM_ResetEvent(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event)

Reset event counters.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to syspm_monitor_t.

void SYSPM_ResetInstructionEvent(SYSPM_Type *base, *syspm_monitor_t* monitor)

Reset Instruction Counter.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to syspm_monitor_t.

void SYSPM_SetCountMode(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_mode_t* mode)

Set count mode.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to syspm_monitor_t.
- mode – syspm select counter mode, see to syspm_mode_t.

void SYSPM_SetStartStopControl(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_startstop_control_t* ssc)

Set Start/Stop Control.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to syspm_monitor_t.
- ssc – This 3-bit field provides a three-phase mechanism to start/stop the counters. It includes a prioritized scheme with local start > local stop > global start > global stop > conditional TSTART > TSTOP. The global and conditional start/stop affect all configured PM/PSAM module concurrently so counters are “coherent”. see to syspm_startstop_control_t

void SYSPM_DisableCounter(SYSPM_Type *base, *syspm_monitor_t* monitor)

Disable Counters if Stopped or Halted.

Parameters

- base – SYSPM peripheral base address.

- monitor – syspm control monitor, see to syspm_monitor_t.

uint64_t SYSPM_GetEventCounter(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event)

This is the the 40-bits of eventx counter. The value in this register increments each time the event selected in PMCRx[SELEVTx] occurs.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to syspm_monitor_t.
- event – syspm select event, see to syspm_event_t.

Returns

- When the return value is not equal to SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE, the return value represents a 40 bits eventx counter.
- When the return value is equal to SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE, the return value represents timeout occured.

uint64_t SYSPM_GetInstructionCounter(SYSPM_Type *base, *syspm_monitor_t* monitor)

This is the the 40-bits of instructionx counter. The value in this register increments each time the CPU count signals occurs.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to syspm_monitor_t.

Returns

- When the return value is not equal to SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE, the return value represents a 40 bits instruction counter.
- When the return value is equal to SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE, the return value represents timeout occured.

EVENT_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM event count stable (0 means wait forever)

INSTRUCTION_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM instruction count stable (0 means wait forever)

FSL_SYSPM_DRIVER_VERSION

SYSPM driver version.

SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE

2.83 TRNG: True Random Number Generator

FSL_TRNG_DRIVER_VERSION

TRNG driver version 2.0.19.

Current version: 2.0.19

Change log:

- version 2.0.19
  - Added support for MCXA and MCXL.

- version 2.0.18
  - TRNG health checks now done in software on RT5xx and RT6xx.
- version 2.0.17
  - Added support for RT700.
- version 2.0.16
  - Added support for Dual oscillator mode.
- version 2.0.15
  - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest reccomended by design team.
- version 2.0.14
  - add support for RW610 and RW612
- version 2.0.13
  - After deepsleep it might return error, added clearing bits in TRNG_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG_GetRandomData(), for some data length it doesn't reloading entropy correctly.
- version 2.0.12
  - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.11
  - Add clearing pending errors in TRNG_Init().
- version 2.0.10
  - Fixed doxygen issues.
- version 2.0.9
  - Fix HIS_CCM metrics issues.
- version 2.0.8
  - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.
- version 2.0.7
  - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
  - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.5
  - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG_Init to restart entropy generation
- Version 2.0.2

- fix MISRA issues

- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

enum _trng_sample_mode

TRNG sample mode. Used by trng_config_t.

Values:

enumerator kTRNG_SampleModeVonNeumann

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeRaw

Use raw data into both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeVonNeumannRaw

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum _trng_clock_mode

TRNG clock mode. Used by trng_config_t.

Values:

enumerator kTRNG_ClockModeRingOscillator

Ring oscillator is used to operate the TRNG (default).

enumerator kTRNG_ClockModeSystem

System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

enum _trng_ring_osc_div

TRNG ring oscillator divide. Used by trng_config_t.

Values:

enumerator kTRNG_RingOscDiv0

Ring oscillator with no divide

enumerator kTRNG_RingOscDiv2

Ring oscillator divided-by-2.

enumerator kTRNG_RingOscDiv4

Ring oscillator divided-by-4.

enumerator kTRNG_RingOscDiv8

Ring oscillator divided-by-8.

enum trng_oscillator_mode_t

TRNG oscillator mode . Used by trng_config_t.

Values:

enumerator kTRNG_SingleOscillatorModeOsc1

Single oscillator mode, using OSC1 (default)

enumerator kTRNG_DualOscillatorMode

Dual oscillator mode

enumerator kTRNG_SingleOscillatorModeOsc2

Single oscillator mode, using OSC2

typedef enum *_trng_sample_mode* trng_sample_mode_t

TRNG sample mode. Used by trng_config_t.

typedef enum *_trng_clock_mode* trng_clock_mode_t

TRNG clock mode. Used by trng_config_t.

typedef enum *_trng_ring_osc_div* trng_ring_osc_div_t

TRNG ring oscillator divide. Used by trng_config_t.

typedef enum *trng_oscillator_mode_t* trng_oscillator_mode_t

TRNG oscillator mode . Used by trng_config_t.

typedef struct *_trng_statistical_check_limit* trng_statistical_check_limit_t

Data structure for definition of statistical check limits. Used by trng_config_t.

typedef struct *_trng_user_config* trng_config_t

Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the TRNG_Init() function. It contains all TRNG configurations.

status_t TRNG_GetDefaultConfig(*trng_config_t* *userConfig)

Initializes the user configuration structure to default values.

This function initializes the configuration structure to default values. The default values are platform dependent.

Parameters

- userConfig – User configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

status_t TRNG_Init(TRNG_Type *base, const *trng_config_t* *userConfig)

Initializes the TRNG.

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

- base – TRNG base address
- userConfig – Pointer to the initialization configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

void TRNG_Deinit(TRNG_Type *base)

Shuts down the TRNG.

This function shuts down the TRNG.

Parameters

- base – TRNG base address.

status_t TRNG_GetRandomData(TRNG_Type *base, void *data, size_t dataSize)

Gets random data.

This function gets random data from the TRNG.

Parameters

- base – TRNG base address.

- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

Returns
random data

struct __trng__statistical__check__limit

#include <fsl_trng.h> Data structure for definition of statistical check limits. Used by trng_config_t.

Public Members

uint32_t maximum

Maximum limit.

int32_t minimum

Minimum limit.

struct __trng__user__config

#include <fsl_trng.h> Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the TRNG_Init() function. It contains all TRNG configurations.

Public Members

bool lock

Disable programmability of TRNG registers.

trng_clock_mode_t clockMode

Clock mode used to operate TRNG.

trng_ring_osc_div_t ringOscDiv

Ring oscillator divide used by TRNG.

trng_sample_mode_t sampleMode

Sample mode of the TRNG ring oscillator.

trng_oscillator_mode_t oscillatorMode

TRNG oscillator mode .

trng_ring_osc_div_t ringOsc2Div

Divider used for Ring oscillator 2.

uint16_t entropyDelay

Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken.

uint16_t sampleSize

Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation.

uint16_t sparseBitLimit

Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated. This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

uint8_t retryCount

Retry count. It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

uint8_t longRunMaxLimit

Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

trng_statistical_check_limit_t monobitLimit

Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.

trng_statistical_check_limit_t runBit1Limit

Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.

trng_statistical_check_limit_t runBit2Limit

Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.

trng_statistical_check_limit_t runBit3Limit

Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.

trng_statistical_check_limit_t runBit4Limit

Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.

trng_statistical_check_limit_t runBit5Limit

Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.

trng_statistical_check_limit_t runBit6PlusLimit

Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.

trng_statistical_check_limit_t pokerLimit

Maximum and minimum limits for statistical check of "Poker Test".

trng_statistical_check_limit_t frequencyCountLimit

Maximum and minimum limits for statistical check of entropy sample frequency count.

2.84 USDHC: Ultra Secured Digital Host Controller Driver

void USDHC_Init(USDHC_Type *base, const *usdhc_config_t* *config)

USDHC module initialization function.

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – USDHC configuration information.

Return values

kStatus_Success – Operate successfully.

void USDHC_Deinit(USDHC_Type *base)

Deinitializes the USDHC.

Parameters

- base – USDHC peripheral base address.

bool USDHC_Reset(USDHC_Type *base, uint32_t mask, uint32_t timeout)

Resets the USDHC.

Parameters

- base – USDHC peripheral base address.
- mask – The reset type mask(_usdhc_reset).
- timeout – Timeout for reset.

Return values

- true – Reset successfully.
- false – Reset failed.

status_t USDHC_SetAdmaTableConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_data_t* *dataConfig, uint32_t flags)

Sets the DMA descriptor table configuration. A high level DMA descriptor configuration function.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration
- dataConfig – Data descriptor
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetInternalDmaConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)

Internal DMA configuration. This function is used to config the USDHC DMA related registers.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration.
- dataAddr – Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
- enAutoCmd23 – Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA2Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA2 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA1Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA1 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

static inline void USDHC_EnableInternalDMA(USDHC_Type *base, bool enable)

Enables internal DMA.

Parameters

- base – USDHC peripheral base address.
- enable – enable or disable flag

static inline void USDHC_EnableInterruptStatus(USDHC_Type *base, uint32_t mask)

Enables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – Interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_DisableInterruptStatus(USDHC_Type *base, uint32_t mask)

Disables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_EnableInterruptSignal(USDHC_Type *base, uint32_t mask)

Enables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_DisableInterruptSignal(USDHC_Type *base, uint32_t mask)

Disables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline uint32_t USDHC_GetEnabledInterruptStatusFlags(USDHC_Type *base)

Gets the enabled interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline uint32_t USDHC_GetInterruptStatusFlags(USDHC_Type *base)

Gets the current interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline void USDHC_ClearInterruptStatusFlags(USDHC_Type *base, uint32_t mask)

Clears a specified interrupt status. write 1 clears.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(_usdhc_interrupt_status_flag).

static inline uint32_t USDHC_GetAutoCommand12ErrorStatusFlags(USDHC_Type *base)

Gets the status of auto command 12 error.

Parameters

- base – USDHC peripheral base address.

Returns

Auto command 12 error status flags mask(_usdhc_auto_command12_error_status_flag).

static inline uint32_t USDHC_GetAdmaErrorStatusFlags(USDHC_Type *base)

Gets the status of the ADMA error.

Parameters

- base – USDHC peripheral base address.

Returns

ADMA error status flags mask(_usdhc_adma_error_status_flag).

static inline uint32_t USDHC_GetPresentStatusFlags(USDHC_Type *base)

Gets a present status.

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

- base – USDHC peripheral base address.

Returns

Present USDHC's status flags mask(_usdhc_present_status_flag).

void USDHC_GetCapability(USDHC_Type *base, *usdhc_capability_t* *capability)

Gets the capability information.

Parameters

- base – USDHC peripheral base address.
- capability – Structure to save capability information.

static inline void USDHC_ForceClockOn(USDHC_Type *base, bool enable)

Forces the card clock on.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

uint32_t USDHC_SetSdClock(USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)

Sets the SD bus clock frequency.

Parameters

- base – USDHC peripheral base address.
- srcClock_Hz – USDHC source clock frequency united in Hz.
- busClock_Hz – SD bus clock frequency united in Hz.

Returns

The nearest frequency of busClock_Hz configured for SD bus.

bool USDHC_SetCardActive(USDHC_Type *base, uint32_t timeout)

Sends 80 clocks to the card to set it to the active state.

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

- base – USDHC peripheral base address.
- timeout – Timeout to initialize card.

Return values

- true – Set card active successfully.
- false – Set card active failed.

static inline void USDHC_AssertHardwareReset(USDHC_Type *base, bool high)

Triggers a hardware reset.

Parameters

- base – USDHC peripheral base address.
- high – 1 or 0 level

static inline void USDHC_SetDataBusWidth(USDHC_Type *base, *usdhc_data_bus_width_t* width)

Sets the data transfer width.

Parameters

- base – USDHC peripheral base address.
- width – Data transfer width.

static inline void USDHC_WriteData(USDHC_Type *base, uint32_t data)

Fills the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.
- data – The data about to be sent.

static inline uint32_t USDHC_ReadData(USDHC_Type *base)

Retrieves the data from the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.

Returns

The data has been read.

void USDHC_SendCommand(USDHC_Type *base, *usdhc_command_t* *command)

Sends command function.

Parameters

- base – USDHC peripheral base address.
- command – configuration

static inline void USDHC_EnableWakeupEvent(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables a wakeup event in low-power mode.

Parameters

- base – USDHC peripheral base address.
- mask – Wakeup events mask(_usdhc_wakeup_event).
- enable – True to enable, false to disable.

static inline void USDHC_CardDetectByData3(USDHC_Type *base, bool enable)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline bool USDHC_DetectCardInsert(USDHC_Type *base)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableSdioControl(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables the SDIO card control.

Parameters

- base – USDHC peripheral base address.
- mask – SDIO card control flags mask(_usdhc_sdio_control_flag).
- enable – True to enable, false to disable.

static inline void USDHC_SetContinueRequest(USDHC_Type *base)

Restarts a transaction which has stopped at the block GAP for the SDIO card.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_RequestStopAtBlockGap(USDHC_Type *base, bool enable)

Request stop at block gap function.

Parameters

- base – USDHC peripheral base address.
- enable – True to stop at block gap, false to normal transfer.

void USDHC_SetMmcBootConfig(USDHC_Type *base, const *usdhc_boot_config_t* *config)

Configures the MMC boot feature.

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – The MMC boot configuration information.

static inline void USDHC_EnableMmcBoot(USDHC_Type *base, bool enable)

Enables or disables the mmc boot mode.

Parameters

- base – USDHC peripheral base address.
- enable – True to enable, false to disable.

static inline void USDHC_SetForceEvent(USDHC_Type *base, uint32_t mask)

Forces generating events according to the given mask.

Parameters

- base – USDHC peripheral base address.

- mask – The force events bit posistion (_usdhc_force_event).

static inline bool USDHC_RequestTuningForSDR50(USDHC_Type *base)

Checks the SDR50 mode request tuning bit. When this bit set, application shall perform tuning for SDR50 mode.

Parameters

- base – USDHC peripheral base address.

static inline bool USDHC_RequestReTuning(USDHC_Type *base)

Checks the request re-tuning bit. When this bit is set, user should do manual tuning or standard tuning function.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableAutoTuning(USDHC_Type *base, bool enable)

The SDR104 mode auto tuning enable and disable. This function should be called after tuning function execute pass, auto tuning will handle by hardware.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC_EnableAutoTuningForCmdAndData(USDHC_Type *base)

The auto tuning enbale for CMD/DATA line.

Parameters

- base – USDHC peripheral base address.

void USDHC_EnableManualTuning(USDHC_Type *base, bool enable)

Manual tuning trigger or abort. User should handle the tuning cmd and find the boundary of the delay then calucate a average value which will be configured to the **CLK_TUNE_CTRL_STATUS** This function should be called before function USDHC_AdjustDelayForManualTuning.

Parameters

- base – USDHC peripheral base address.
- enable – tuning enable flag

static inline uint32_t USDHC_GetTuningDelayStatus(USDHC_Type *base)

Get the tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.

Return values

CLK – Tuning Control and Status register value.

status_t USDHC_SetTuningDelay(USDHC_Type *base, uint32_t preDelay, uint32_t outDelay, uint32_t postDelay)

The tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.
- preDelay – Set the number of delay cells on the feedback clock between the feedback clock and CLK_PRE.

- outDelay – Set the number of delay cells on the feedback clock between CLK_PRE and CLK_OUT.
- postDelay – Set the number of delay cells on the feedback clock between CLK_OUT and CLK_POST.

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

status_t USDHC_AdjustDelayForManualTuning(USDHC_Type *base, uint32_t delay)

Adjusts delay for mannual tuning.

Deprecated:

Do not use this function. It has been superceded by USDHC_SetTuingDelay

Parameters

- base – USDHC peripheral base address.
- delay – setting configuration

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

static inline void USDHC_SetStandardTuningCounter(USDHC_Type *base, uint8_t counter)

set tuning counter tuning.

Parameters

- base – USDHC peripheral base address.
- counter – tuning counter

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

void USDHC_EnableStandardTuning(USDHC_Type *base, uint32_t tuningStartTap, uint32_t step, bool enable)

The enable standard tuning function. The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

Parameters

- base – USDHC peripheral base address.
- tuningStartTap – start tap
- step – tuning step
- enable – enable/disable flag

static inline uint32_t USDHC_GetExecuteStdTuningStatus(USDHC_Type *base)

Gets execute STD tuning status.

Parameters

- base – USDHC peripheral base address.

static inline uint32_t USDHC_CheckStdTuningResult(USDHC_Type *base)

Checks STD tuning result.

Parameters

- base – USDHC peripheral base address.

static inline uint32_t USDHC_CheckTuningError(USDHC_Type *base)

Checks tuning error.

Parameters

- base – USDHC peripheral base address.

void USDHC_EnableDDRMode(USDHC_Type *base, bool enable, uint32_t nibblePos)

The enable/disable DDR mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag
- nibblePos – nibble position

static inline void USDHC_EnableHS400Mode(USDHC_Type *base, bool enable)

The enable/disable HS400 mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline void USDHC_ResetStrobeDLL(USDHC_Type *base)

Resets the strobe DLL.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableStrobeDLL(USDHC_Type *base, bool enable)

Enables/disables the strobe DLL.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC_ConfigStrobeDLL(USDHC_Type *base, uint32_t delayTarget, uint32_t updateInterval)

Configs the strobe DLL delay target and update interval.

Parameters

- base – USDHC peripheral base address.
- delayTarget – delay target
- updateInterval – update interval

static inline void USDHC_SetStrobeDllOverride(USDHC_Type *base, uint32_t delayTaps)

Enables manual override for slave delay chain using **STROBE_SLV_OVERRIDE_VAL**.

Parameters

- base – USDHC peripheral base address.
- delayTaps – Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

static inline uint32_t USDHC_GetStrobeDLLStatus(USDHC_Type *base)

Gets the strobe DLL status.

Parameters

- base – USDHC peripheral base address.

void USDHC_SetDataConfig(USDHC_Type *base, *usdhc_transfer_direction_t* dataDirection, uint32_t blockCount, uint32_t blockSize)

USDHC data configuration.

Parameters

- base – USDHC peripheral base address.
- dataDirection – Data direction, tx or rx.
- blockCount – Data block count.
- blockSize – Data block size.

void USDHC_TransferCreateHandle(USDHC_Type *base, *usdhc_handle_t* *handle, const *usdhc_transfer_callback_t* *callback, void *userData)

Creates the USDHC handle.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle pointer.
- callback – Structure pointer to contain all callback functions.
- userData – Callback function parameter.

status_t USDHC_TransferNonBlocking(USDHC_Type *base, *usdhc_handle_t* *handle, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using an interrupt and an asynchronous method.

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note: Call API USDHC_TransferCreateHandle when calling this API.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.
- dmaConfig – ADMA configuration.
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_BusyTransferring – Busy transferring.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_Success – Operate successfully.

status_t USDHC_TransferBlocking(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using a blocking method.

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note: There is no need to call API USDHC_TransferCreateHandle when calling this API.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – adma configuration
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_USDHC_SendCommandFailed – Send command failed.
- kStatus_USDHC_TransferDataFailed – Transfer data failed.
- kStatus_Success – Operate successfully.

void USDHC_TransferHandleIRQ(USDHC_Type *base, *usdhc_handle_t* *handle)

IRQ handler for the USDHC.

This function deals with the IRQs on the given host controller.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.

FSL_USDHC_DRIVER_VERSION

Driver version 2.8.5.

Enum _usdhc_status. USDHC status.

Values:

enumerator kStatus_USDHC_BusyTransferring

Transfer is on-going.

enumerator kStatus_USDHC_PrepareAdmaDescriptorFailed

Set DMA descriptor failed.

enumerator kStatus_USDHC_SendCommandFailed

Send command failed.

enumerator kStatus_USDHC_TransferDataFailed

Transfer data failed.

enumerator kStatus_USDHC_DMADataAddrNotAlign
Data address not aligned.

enumerator kStatus_USDHC_ReTuningRequest
Re-tuning request.

enumerator kStatus_USDHC_TuningError
Tuning error.

enumerator kStatus_USDHC_NotSupport
Not support.

enumerator kStatus_USDHC_TransferDataComplete
Transfer data complete.

enumerator kStatus_USDHC_SendCommandSuccess
Transfer command complete.

enumerator kStatus_USDHC_TransferDMAComplete
Transfer DMA complete.

Enum _usdhc_capability_flag. Host controller capabilities flag mask. .

Values:

enumerator kUSDHC_SupportAdmaFlag
Support ADMA.

enumerator kUSDHC_SupportHighSpeedFlag
Support high-speed.

enumerator kUSDHC_SupportDmaFlag
Support DMA.

enumerator kUSDHC_SupportSuspendResumeFlag
Support suspend/resume.

enumerator kUSDHC_SupportV330Flag
Support voltage 3.3V.

enumerator kUSDHC_SupportV300Flag
Support voltage 3.0V.

enumerator kUSDHC_Support4BitFlag
Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.

enumerator kUSDHC_Support8BitFlag
Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.

enumerator kUSDHC_SupportDDR50Flag
SD version 3.0 new feature, supporting DDR50 mode.

enumerator kUSDHC_SupportSDR104Flag
Support SDR104 mode.

enumerator kUSDHC_SupportSDR50Flag
Support SDR50 mode.

Enum _usdhc_wakeup_event. Wakeup event mask. .

Values:

enumerator kUSDHC_WakeupEventOnCardInt
Wakeup on card interrupt.

enumerator kUSDHC_WakeupEventOnCardInsert
Wakeup on card insertion.

enumerator kUSDHC_WakeupEventOnCardRemove
Wakeup on card removal.

enumerator kUSDHC_WakeupEventsAll
All wakeup events

Enum _usdhc_reset. Reset type mask. .

Values:

enumerator kUSDHC_ResetAll
Reset all except card detection.

enumerator kUSDHC_ResetCommand
Reset command line.

enumerator kUSDHC_ResetData
Reset data line.

enumerator kUSDHC_ResetTuning
Reset tuning circuit.

enumerator kUSDHC_ResetsAll
All reset types

Enum _usdhc_transfer_flag. Transfer flag mask.

Values:

enumerator kUSDHC_EnableDmaFlag
Enable DMA.

enumerator kUSDHC_CommandTypeSuspendFlag
Suspend command.

enumerator kUSDHC_CommandTypeResumeFlag
Resume command.

enumerator kUSDHC_CommandTypeAbortFlag
Abort command.

enumerator kUSDHC_EnableBlockCountFlag
Enable block count.

enumerator kUSDHC_EnableAutoCommand12Flag
Enable auto CMD12.

enumerator kUSDHC_DataReadFlag
Enable data read.

enumerator kUSDHC_MultipleBlockFlag
Multiple block data read/write.

enumerator kUSDHC_EnableAutoCommand23Flag
Enable auto CMD23.

enumerator kUSDHC_ResponseLength136Flag
136-bit response length.

enumerator kUSDHC_ResponseLength48Flag
48-bit response length.

enumerator kUSDHC_ResponseLength48BusyFlag
48-bit response length with busy status.

enumerator kUSDHC_EnableCrcCheckFlag
Enable CRC check.

enumerator kUSDHC_EnableIndexCheckFlag
Enable index check.

enumerator kUSDHC_DataPresentFlag
Data present flag.

Enum _usdhc_present_status_flag. Present status flag mask. .

Values:

enumerator kUSDHC_CommandInhibitFlag
Command inhibit.

enumerator kUSDHC_DataInhibitFlag
Data inhibit.

enumerator kUSDHC_DataLineActiveFlag
Data line active.

enumerator kUSDHC_SdClockStableFlag
SD bus clock stable.

enumerator kUSDHC_WriteTransferActiveFlag
Write transfer active.

enumerator kUSDHC_ReadTransferActiveFlag
Read transfer active.

enumerator kUSDHC_BufferWriteEnableFlag
Buffer write enable.

enumerator kUSDHC_BufferReadEnableFlag
Buffer read enable.

enumerator kUSDHC_ReTuningRequestFlag
Re-tuning request flag, only used for SDR104 mode.

enumerator kUSDHC_DelaySettingFinishedFlag
Delay setting finished flag.

enumerator kUSDHC_CardInsertedFlag
Card inserted.

enumerator kUSDHC_CommandLineLevelFlag
Command line signal level.

enumerator kUSDHC_Data0LineLevelFlag
Data0 line signal level.

enumerator kUSDHC_Data1LineLevelFlag
Data1 line signal level.

enumerator kUSDHC_Data2LineLevelFlag
Data2 line signal level.

enumerator kUSDHC_Data3LineLevelFlag
Data3 line signal level.

enumerator kUSDHC_Data4LineLevelFlag
Data4 line signal level.

enumerator kUSDHC_Data5LineLevelFlag
Data5 line signal level.

enumerator kUSDHC_Data6LineLevelFlag
Data6 line signal level.

enumerator kUSDHC_Data7LineLevelFlag
Data7 line signal level.

Enum _usdhc_interrupt_status_flag. Interrupt status flag mask. .

Values:

enumerator kUSDHC_CommandCompleteFlag
Command complete.

enumerator kUSDHC_DataCompleteFlag
Data complete.

enumerator kUSDHC_BlockGapEventFlag
Block gap event.

enumerator kUSDHC_DmaCompleteFlag
DMA interrupt.

enumerator kUSDHC_BufferWriteReadyFlag
Buffer write ready.

enumerator kUSDHC_BufferReadReadyFlag
Buffer read ready.

enumerator kUSDHC_CardInsertionFlag
Card inserted.

enumerator kUSDHC_CardRemovalFlag
Card removed.

enumerator kUSDHC_CardInterruptFlag
Card interrupt.

enumerator kUSDHC_ReTuningEventFlag
Re-Tuning event, only for SD3.0 SDR104 mode.

enumerator kUSDHC_TuningPassFlag
SDR104 mode tuning pass flag.

enumerator kUSDHC_TuningErrorFlag
SDR104 tuning error flag.

enumerator kUSDHC_CommandTimeoutFlag
Command timeout error.

enumerator kUSDHC_CommandCrcErrorFlag
Command CRC error.

enumerator kUSDHC_CommandEndBitErrorFlag
Command end bit error.

enumerator kUSDHC_CommandIndexErrorFlag
Command index error.

enumerator kUSDHC_DataTimeoutFlag
Data timeout error.

enumerator kUSDHC_DataCrcErrorFlag
Data CRC error.

enumerator kUSDHC_DataEndBitErrorFlag
Data end bit error.

enumerator kUSDHC_AutoCommand12ErrorFlag
Auto CMD12 error.

enumerator kUSDHC_DmaErrorFlag
DMA error.

enumerator kUSDHC_CommandErrorFlag
Command error

enumerator kUSDHC_DataErrorFlag
Data error

enumerator kUSDHC_ErrorFlag
All error

enumerator kUSDHC_DataFlag
Data interrupts

enumerator kUSDHC_DataDMAFlag
Data interrupts

enumerator kUSDHC_CommandFlag
Command interrupts

enumerator kUSDHC_CardDetectFlag
Card detection interrupts

enumerator kUSDHC_SDR104TuningFlag
SDR104 tuning flag.

enumerator kUSDHC_AllInterruptFlags
All flags mask

Enum _usdhc_auto_command12_error_status_flag. Auto CMD12 error status flag mask. .

Values:

enumerator kUSDHC_AutoCommand12NotExecutedFlag
Not executed error.

enumerator kUSDHC_AutoCommand12TimeoutFlag
Timeout error.

enumerator kUSDHC_AutoCommand12EndBitErrorFlag
End bit error.

enumerator kUSDHC_AutoCommand12CrcErrorFlag
CRC error.

enumerator kUSDHC_AutoCommand12IndexErrorFlag
Index error.

enumerator kUSDHC_AutoCommand12NotIssuedFlag
Not issued error.

Enum _usdhc_standard_tuning. Standard tuning flag.

Values:

enumerator kUSDHC_ExecuteTuning
Used to start tuning procedure.

enumerator kUSDHC_TuningSampleClockSel
When **std_tuning_en** bit is set, this bit is used to select sampleing clock.

Enum _usdhc_adma_error_status_flag. ADMA error status flag mask. .

Values:

enumerator kUSDHC_AdmaLenghMismatchFlag
Length mismatch error.

enumerator kUSDHC_AdmaDescriptorErrorFlag
Descriptor error.

Enum _usdhc_adma_error_state. ADMA error state.

This state is the detail state when ADMA error has occurred.

Values:

enumerator kUSDHC_AdmaErrorStateStopDma
Stop DMA, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateFetchDescriptor
Fetch descriptor, current location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateChangeAddress
Change address, no DMA error has occurred.

enumerator kUSDHC_AdmaErrorStateTransferData
Transfer data, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateInvalidLength
Invalid length in ADMA descriptor.

enumerator kUSDHC_AdmaErrorStateInvalidDescriptor
Invalid descriptor fetched by ADMA.

enumerator kUSDHC_AdmaErrorState
ADMA error state

Enum _usdhc_force_event. Force event bit position. .

Values:

enumerator kUSDHC_ForceEventAutoCommand12NotExecuted
Auto CMD12 not executed error.

enumerator kUSDHC_ForceEventAutoCommand12Timeout
Auto CMD12 timeout error.

enumerator kUSDHC_ForceEventAutoCommand12CrcError
Auto CMD12 CRC error.

enumerator kUSDHC_ForceEventEndBitError
Auto CMD12 end bit error.

enumerator kUSDHC_ForceEventAutoCommand12IndexError
Auto CMD12 index error.

enumerator kUSDHC_ForceEventAutoCommand12NotIssued
Auto CMD12 not issued error.

enumerator kUSDHC_ForceEventCommandTimeout
Command timeout error.

enumerator kUSDHC_ForceEventCommandCrcError
Command CRC error.

enumerator kUSDHC_ForceEventCommandEndBitError
Command end bit error.

enumerator kUSDHC_ForceEventCommandIndexError
Command index error.

enumerator kUSDHC_ForceEventDataTimeout
Data timeout error.

enumerator kUSDHC_ForceEventDataCrcError
Data CRC error.

enumerator kUSDHC_ForceEventDataEndBitError
Data end bit error.

enumerator kUSDHC_ForceEventAutoCommand12Error
Auto CMD12 error.

enumerator kUSDHC_ForceEventCardInt
Card interrupt.

enumerator kUSDHC_ForceEventDmaError
Dma error.

enumerator kUSDHC_ForceEventTuningError
Tuning error.

enumerator kUSDHC_ForceEventsAll
All force event flags mask.

enum __usdhc_transfer_direction
Data transfer direction.

Values:

enumerator kUSDHC_TransferDirectionReceive
USDHC transfer direction receive.

enumerator kUSDHC_TransferDirectionSend
USDHC transfer direction send.

enum _usdhc_data_bus_width
Data transfer width.

Values:

enumerator kUSDHC_DataBusWidth1Bit
1-bit mode

enumerator kUSDHC_DataBusWidth4Bit
4-bit mode

enumerator kUSDHC_DataBusWidth8Bit
8-bit mode

enum _usdhc_endian_mode
Endian mode.

Values:

enumerator kUSDHC_EndianModeBig
Big endian mode.

enumerator kUSDHC_EndianModeHalfWordBig
Half word big endian mode.

enumerator kUSDHC_EndianModeLittle
Little endian mode.

enum _usdhc_dma_mode
DMA mode.

Values:

enumerator kUSDHC_DmaModeSimple
External DMA.

enumerator kUSDHC_DmaModeAdma1
ADMA1 is selected.

enumerator kUSDHC_DmaModeAdma2
ADMA2 is selected.

enumerator kUSDHC_ExternalDMA
External DMA mode selected.

Enum _usdhc_sdio_control_flag. SDIO control flag mask. .

Values:

enumerator kUSDHC_StopAtBlockGapFlag
Stop at block gap.

enumerator kUSDHC_ReadWaitControlFlag
Read wait control.

enumerator kUSDHC_InterruptAtBlockGapFlag
Interrupt at block gap.

enumerator kUSDHC_ReadDoneNo8CLK
Read done without 8 clk for block gap.

enumerator kUSDHC_ExactBlockNumberReadFlag
Exact block number read.

enum _usdhc_boot_mode
MMC card boot mode.

Values:

enumerator kUSDHC_BootModeNormal
Normal boot

enumerator kUSDHC_BootModeAlternative
Alternative boot

enum _usdhc_card_command_type
The command type.

Values:

enumerator kCARD_CommandTypeNormal
Normal command

enumerator kCARD_CommandTypeSuspend
Suspend command

enumerator kCARD_CommandTypeResume
Resume command

enumerator kCARD_CommandTypeAbort
Abort command

enumerator kCARD_CommandTypeEmpty
Empty command

enum _usdhc_card_response_type
The command response type.

Defines the command response type from card to host controller.

Values:

enumerator kCARD_ResponseTypeNone
Response type: none

enumerator kCARD_ResponseTypeR1
Response type: R1

enumerator kCARD_ResponseTypeR1b
Response type: R1b

enumerator kCARD_ResponseTypeR2
Response type: R2

enumerator kCARD_ResponseTypeR3
Response type: R3

enumerator kCARD_ResponseTypeR4
Response type: R4

enumerator kCARD_ResponseTypeR5
Response type: R5

enumerator kCARD_ResponseTypeR5b
Response type: R5b

enumerator kCARD_ResponseTypeR6
Response type: R6

enumerator kCARD_ResponseTypeR7
Response type: R7

Enum _usdhc_adma1_descriptor_flag. The mask for the control/status field in ADMA1 descriptor.

Values:

enumerator kUSDHC_Adma1DescriptorValidFlag
Valid flag.

enumerator kUSDHC_Adma1DescriptorEndFlag
End flag.

enumerator kUSDHC_Adma1DescriptorInterrupFlag
Interrupt flag.

enumerator kUSDHC_Adma1DescriptorActivity1Flag
Activity 1 flag.

enumerator kUSDHC_Adma1DescriptorActivity2Flag
Activity 2 flag.

enumerator kUSDHC_Adma1DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma1DescriptorTypeTransfer
Transfer data.

enumerator kUSDHC_Adma1DescriptorTypeLink
Link descriptor.

enumerator kUSDHC_Adma1DescriptorTypeSetLength
Set data length.

Enum _usdhc_adma2_descriptor_flag. ADMA1 descriptor control and status mask.

Values:

enumerator kUSDHC_Adma2DescriptorValidFlag
Valid flag.

enumerator kUSDHC_Adma2DescriptorEndFlag
End flag.

enumerator kUSDHC_Adma2DescriptorInterruptFlag
Interrupt flag.

enumerator kUSDHC_Adma2DescriptorActivity1Flag
Activity 1 mask.

enumerator kUSDHC_Adma2DescriptorActivity2Flag
Activity 2 mask.

enumerator kUSDHC_Adma2DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma2DescriptorTypeReserved
Reserved.

enumerator kUSDHC_Adma2DescriptorTypeTransfer
Transfer type.

enumerator kUSDHC_Adma2DescriptorTypeLink
Link type.

Enum _usdhc_adma_flag. ADMA descriptor configuration flag. .

Values:

enumerator kUSDHC_AdmaDescriptorSingleFlag
Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

enumerator kUSDHC_AdmaDescriptorMultipleFlag
Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

enum _usdhc_burst_len
DMA transfer burst len config.

Values:

enumerator kUSDHC_EnBurstLenForINCR
Enable burst len for INCR.

enumerator kUSDHC_EnBurstLenForINCR4816
Enable burst len for INCR4/INCR8/INCR16.

enumerator kUSDHC_EnBurstLenForINCR4816WRAP
Enable burst len for INCR4/8/16 WRAP.

Enum _usdhc_transfer_data_type. Tansfer data type definition.

Values:

enumerator kUSDHC_TransferDataNormal
Transfer normal read/write data.

enumerator kUSDHC_TransferDataTuning
Transfer tuning data.

enumerator kUSDHC_TransferDataBoot
Transfer boot data.

enumerator kUSDHC_TransferDataBootcontinous
Transfer boot data continuously.

typedef enum *_usdhc_transfer_direction* usdhc_transfer_direction_t
Data transfer direction.

typedef enum *_usdhc_data_bus_width* usdhc_data_bus_width_t
Data transfer width.

typedef enum *_usdhc_endian_mode* usdhc_endian_mode_t
Endian mode.

typedef enum *_usdhc_dma_mode* usdhc_dma_mode_t
DMA mode.

typedef enum *_usdhc_boot_mode* usdhc_boot_mode_t
MMC card boot mode.

typedef enum *_usdhc_card_command_type* usdhc_card_command_type_t
The command type.

typedef enum *_usdhc_card_response_type* usdhc_card_response_type_t
The command response type.

Defines the command response type from card to host controller.

typedef enum *_usdhc_burst_len* usdhc_burst_len_t
DMA transfer burst len config.

typedef uint32_t usdhc_adma1_descriptor_t
Defines the ADMA1 descriptor structure.

typedef struct *_usdhc_adma2_descriptor* usdhc_adma2_descriptor_t
Defines the ADMA2 descriptor structure.

typedef struct *_usdhc_capability* usdhc_capability_t
USDHC capability information.

Defines a structure to save the capability information of USDHC.

typedef struct *_usdhc_boot_config* usdhc_boot_config_t
Data structure to configure the MMC boot feature.

typedef struct *_usdhc_config* usdhc_config_t
Data structure to initialize the USDHC.

typedef struct *_usdhc_command* usdhc_command_t
Card command descriptor.

Defines card command-related attribute.

typedef struct *_usdhc_adma_config* usdhc_adma_config_t
ADMA configuration.

typedef struct *_usdhc_scatter_gather_data_list* usdhc_scatter_gather_data_list_t
Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

typedef struct *_usdhc_scatter_gather_data* usdhc_scatter_gather_data_t
Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_scatter_gather_transfer* usdhc_scatter_gather_transfer_t
usdhc scatter gather transfer.

typedef struct *_usdhc_data* usdhc_data_t
Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_transfer* usdhc_transfer_t
Transfer state.

typedef struct *_usdhc_handle* usdhc_handle_t
USDHC handle typedef.

typedef struct *_usdhc_transfer_callback* usdhc_transfer_callback_t
USDHC callback functions.

typedef *status_t* (*usdhc_transfer_function_t)(USDHC_Type *base, *usdhc_transfer_t* *content)
USDHC transfer function.

typedef struct *_usdhc_host* usdhc_host_t
USDHC host descriptor.

USDHC_MAX_BLOCK_COUNT
Maximum block count can be set one time.

FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER
USDHC scatter gather feature control macro.

USDHC_ADMA1_ADDRESS_ALIGN
The alignment size for ADDRESS filed in ADMA1's descriptor.

USDHC_ADMA1_LENGTH_ALIGN
The alignment size for LENGTH field in ADMA1's descriptor.

USDHC_ADMA2_ADDRESS_ALIGN
The alignment size for ADDRESS field in ADMA2's descriptor.

USDHC_ADMA2_LENGTH_ALIGN
The alignment size for LENGTH filed in ADMA2's descriptor.

USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT
The bit shift for ADDRESS filed in ADMA1's descriptor.

Address/page field	Reserved	Attribute					
31 12	11 6	05	04	03	02	01	00
address or data length	000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don't care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK
The bit mask for ADDRESS field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT
The bit shift for LENGTH filed in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK
The mask for LENGTH field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH filed in ADMA1's descriptor. Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH field in ADMA2's descriptor.

Address field	Length	Reserved	Attribute					
63 32	31 16	15 06	05	04	03	02	01	00
32-bit address	16-bit length	0000000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	Operation
0	0	No op	Don't care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK

The bit mask for LENGTH field in ADMA2's descriptor.

USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH field in ADMA2's descriptor.

struct _usdhc_adma2_descriptor

#include <fsl_usdhc.h> Defines the ADMA2 descriptor structure.

Public Members

uint32_t attribute

The control and status field.

uint32_t address

The address field.

struct _usdhc_capability

#include <fsl_usdhc.h> USDHC capability information.

Defines a structure to save the capability information of USDHC.

Public Members

uint32_t sdVersion

Support SD card/sdio version.

uint32_t mmcVersion

Support EMMC card version.

uint32_t maxBlockLength
: Maximum block length united as byte.

uint32_t maxBlockCount
: Maximum block count can be set one time.

uint32_t flags
: Capability flags to indicate the support information(_usdhc_capability_flag).

struct _usdhc_boot_config
: *#include <fsl_usdhc.h>* Data structure to configure the MMC boot feature.

Public Members

uint32_t ackTimeoutCount
: Timeout value for the boot ACK. The available range is 0 ~ 15.

usdhc_boot_mode_t bootMode
: Boot mode selection.

uint32_t blockCount
: Stop at block gap value of automatic mode. Available range is 0 ~ 65535.

size_t blockSize
: Block size.

bool enableBootAck
: Enable or disable boot ACK.

bool enableAutoStopAtBlockGap
: Enable or disable auto stop at block gap function in boot period.

struct _usdhc_config
: *#include <fsl_usdhc.h>* Data structure to initialize the USDHC.

Public Members

uint32_t dataTimeout
: Data timeout value.

usdhc_endian_mode_t endianMode
: Endian mode.

uint8_t readWatermarkLevel
: Watermark level for DMA read operation. Available range is 1 ~ 128.

uint8_t writeWatermarkLevel
: Watermark level for DMA write operation. Available range is 1 ~ 128.

struct _usdhc_command
: *#include <fsl_usdhc.h>* Card command descriptor.

Defines card command-related attribute.

Public Members

uint32_t index
: Command index.

uint32_t argument
: Command argument.

usdhc_card_command_type_t type
: Command type.

usdhc_card_response_type_t responseType
: Command response type.

uint32_t response[4U]
: Response for this command.

uint32_t responseErrorFlags
: Response error flag, which need to check the command reponse.

uint32_t flags
: Cmd flags.

struct __usdhc_adma_config
: *#include <fsl_usdhc.h>* ADMA configuration.

Public Members

usdhc_dma_mode_t dmaMode
: DMA mode.

uint32_t *admaTable
: ADMA table address, can't be null if transfer way is ADMA1/ADMA2.

uint32_t admaTableWords
: ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

struct __usdhc_scatter_gather_data_list
: *#include <fsl_usdhc.h>* Card scatter gather data list.

  Allow application register uncontinuous data buffer for data transfer.

struct __usdhc_scatter_gather_data
: *#include <fsl_usdhc.h>* Card scatter gather data descriptor.

  Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12
: Enable auto CMD12.

bool enableAutoCommand23
: Enable auto CMD23.

bool enableIgnoreError
: Enable to ignore error event to read/write all the data.

usdhc_transfer_direction_t dataDirection
: data direction

uint8_t dataType
: this is used to distinguish the normal/tuning/boot data.

size_t blockSize
: Block size.

usdhc_scatter_gather_data_list_t sgData
: scatter gather data

struct __usdhc__scatter__gather__transfer
: *#include <fsl_usdhc.h>* usdhc scatter gather transfer.

Public Members

usdhc_scatter_gather_data_t *data
: Data to transfer.

usdhc_command_t *command
: Command to send.

struct __usdhc__data
: *#include <fsl_usdhc.h>* Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12
: Enable auto CMD12.

bool enableAutoCommand23
: Enable auto CMD23.

bool enableIgnoreError
: Enable to ignore error event to read/write all the data.

uint8_t dataType
: this is used to distinguish the normal/tuning/boot data.

size_t blockSize
: Block size.

uint32_t blockCount
: Block count.

uint32_t *rxData
: Buffer to save data read.

const uint32_t *txData
: Data buffer to write.

struct __usdhc__transfer
: *#include <fsl_usdhc.h>* Transfer state.

Public Members

usdhc_data_t *data
: Data to transfer.

usdhc_command_t *command
Command to send.

struct __usdhc_transfer_callback
#include <fsl_usdhc.h> USDHC callback functions.

Public Members

void (*CardInserted)(USDHC_Type *base, void *userData)
Card inserted occurs when DAT3/CD pin is for card detect

void (*CardRemoved)(USDHC_Type *base, void *userData)
Card removed occurs

void (*SdioInterrupt)(USDHC_Type *base, void *userData)
SDIO card interrupt occurs

void (*BlockGap)(USDHC_Type *base, void *userData)
stopped at block gap event

void (*TransferComplete)(USDHC_Type *base, *usdhc_handle_t* *handle, *status_t* status, void *userData)
Transfer complete callback.

void (*ReTuning)(USDHC_Type *base, void *userData)
Handle the re-tuning.

struct __usdhc_handle
#include <fsl_usdhc.h> USDHC handle.

Defines the structure to save the USDHC state information and callback function.

Note: All the fields except interruptFlags and transferredWords must be allocated by the user.

Public Members

usdhc_data_t *volatile data
Transfer parameter. Data to transfer.

usdhc_command_t *volatile command
Transfer parameter. Command to send.

volatile uint32_t transferredWords
Transfer status. Words transferred by DATAPORT way.

usdhc_transfer_callback_t callback
Callback function.

void *userData
Parameter for transfer complete callback.

struct __usdhc_host
#include <fsl_usdhc.h> USDHC host descriptor.

Public Members

USDHC_Type *base
: USDHC peripheral base address.

uint32_t sourceClock_Hz
: USDHC source clock frequency united in Hz.

usdhc_config_t config
: USDHC configuration.

usdhc_capability_t capability
: USDHC capability information.

usdhc_transfer_function_t transfer
: USDHC transfer function.

2.85 UTICK: MictoTick Timer Driver

void UTICK_Init(UTICK_Type *base)
: Initializes an UTICK by turning its bus clock on.

void UTICK_Deinit(UTICK_Type *base)
: Deinitializes a UTICK instance.

  This function shuts down Utick bus clock

  Parameters

  - base – UTICK peripheral base address.

uint32_t UTICK_GetStatusFlags(UTICK_Type *base)
: Get Status Flags.

  This returns the status flag

  Parameters

  - base – UTICK peripheral base address.

  Returns
  status register value

void UTICK_ClearStatusFlags(UTICK_Type *base)
: Clear Status Interrupt Flags.

  This clears intr status flag

  Parameters

  - base – UTICK peripheral base address.

  Returns
  none

void UTICK_SetTick(UTICK_Type *base, *utick_mode_t* mode, uint32_t count, *utick_callback_t* cb)
: Starts UTICK.

  This function starts a repeat/onetime countdown with an optional callback

  Parameters

  - base – UTICK peripheral base address.

- mode – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
- count – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
- cb – UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

Returns
none

void UTICK_HandleIRQ(UTICK_Type *base, *utick_callback_t* cb)

UTICK Interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in UTICK_SetTick()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – UTICK peripheral base address.
- cb – callback scheduled for this instance of UTICK

Returns
none

FSL_UTICK_DRIVER_VERSION

UTICK driver version 2.0.5.

enum _utick_mode

UTICK timer operational mode.

Values:

enumerator kUTICK_Onetime

Trigger once

enumerator kUTICK_Repeat

Trigger repeatedly

typedef enum *_utick_mode* utick_mode_t

UTICK timer operational mode.

typedef void (*utick_callback_t)(void)

UTICK callback function.

2.86 WWDT: Windowed Watchdog Timer Driver

void WWDT_GetDefaultConfig(*wwdt_config_t* *config)

Initializes WWDT configure structure.

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

```
config->enableWwdt = true;
config->enableWatchdogReset = false;
config->enableWatchdogProtect = false;
config->enableLockOscillator = false;
config->windowValue = 0xFFFFFFU;
config->timeoutValue = 0xFFFFFFU;
config->warningValue = 0;
```

See also:

wwdt_config_t

Parameters

- config – Pointer to WWDT config structure.

void WWDT_Init(WWDT_Type *base, const *wwdt_config_t* *config)

Initializes the WWDT.

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
wwdt_config_t config;
WWDT_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
WWDT_Init(wwdt_base,&config);
```

Parameters

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

void WWDT_Deinit(WWDT_Type *base)

Shuts down the WWDT.

This function shuts down the WWDT.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Enable(WWDT_Type *base)

Enables the WWDT module.

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Disable(WWDT_Type *base)

Disables the WWDT module.

Deprecated:

Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

- base – WWDT peripheral base address

static inline uint32_t WWDT_GetStatusFlags(WWDT_Type *base)

Gets all WWDT status flags.

This function gets all status flags.

Example for getting Timeout Flag:

```
uint32_t status;
status = WWDT_GetStatusFlags(wwdt_base) & kWWDT_TimeoutFlag;
```

Parameters

- base – WWDT peripheral base address

Returns

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

void WWDT_ClearStatusFlags(WWDT_Type *base, uint32_t mask)

Clear WWDT flag.

This function clears WWDT status flag.

Example for clearing warning flag:

```
WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
```

Parameters

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

static inline void WWDT_SetWarningValue(WWDT_Type *base, uint32_t warningValue)

Set the WWDT warning value.

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

- base – WWDT peripheral base address
- warningValue – WWDT warning value.

static inline void WWDT_SetTimeoutValue(WWDT_Type *base, uint32_t timeoutCount)

Set the WWDT timeout value.

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is TWDCLK*256*4. If enableWatchdogProtect flag is true in wwdt_config_t config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

- base – WWDT peripheral base address
- timeoutCount – WWDT timeout value, count of WWDT clock tick.

static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)

Sets the WWDT window value.

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

- base – WWDT peripheral base address

- windowValue – WWDT window value.

void WWDT_Refresh(WWDT_Type *base)

Refreshes the WWDT timer.

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WWDT peripheral base address

FSL_WWDT_DRIVER_VERSION

Defines WWDT driver version.

WWDT_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WWDT_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

enum _wwdt_status_flags_t

WWDT status flags.

This structure contains the WWDT status flags for use in the WWDT functions.

Values:

enumerator kWWDT_TimeoutFlag

Time-out flag, set when the timer times out

enumerator kWWDT_WarningFlag

Warning interrupt flag, set when timer is below the value WDWARNINT

typedef struct *_wwdt_config* wwdt_config_t

Describes WWDT configuration structure.

struct _wwdt_config

#include <fsl_wwdt.h> Describes WWDT configuration structure.

Public Members

bool enableWwdt

Enables or disables WWDT

bool enableWatchdogReset

true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset

bool enableWatchdogProtect

true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time

uint32_t windowValue

Window value, set this to 0xFFFFFF if windowing is not in effect

uint32_t timeoutValue

Timeout value

uint32_t warningValue

Watchdog time counter value that will generate a warning interrupt. Set this to 0 for no warning

uint32_t clockFreq_Hz
Watchdog clock source frequency.

2.87 CACHE: CACHE Memory Controller

uint32_t XCACHE_GetInstanceByAddr(uint32_t address)
brief Returns an instance number given physical memory address.

param address The physical memory address.

Returns
XCACHE instance number starting from 0.

void XCACHE_EnableCache(XCACHE_Type *base)
Enables the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_DisableCache(XCACHE_Type *base)
Disables the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_InvalidateCache(XCACHE_Type *base)
Invalidates the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_InvalidateCacheByRange(uint32_t address, uint32_t size_byte)
Invalidates cache by range.

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

Parameters

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

void XCACHE_CleanCache(XCACHE_Type *base)
Cleans the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_CleanCacheByRange(uint32_t address, uint32_t size_byte)
Cleans cache by range.

Note: Address and size should be aligned to "XCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to XCACHE_LINESIZE_BYTE if startAddr is not aligned. For the

size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

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

void XCACHE_CleanInvalidateCache(XCACHE_Type *base)

Cleans and invalidates the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate cache by range.

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

Parameters

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

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

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

Parameters

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

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

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

Parameters

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

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

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

Parameters

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

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

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

Parameters

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

FSL_CACHE_DRIVER_VERSION

cache driver version.

XCACHE_LINESIZE_BYTE

cache line size.

2.88 XSPI: Serial Peripheral Interface Driver

Status structure of XSPI.

Values:

enumerator kStatus_XSPI_Busy

XSPI is Busy

enumerator kStatus_XSPI_IpCommandUsageError

XSPI Ip Command Usage Error

enumerator kStatus_XSPI_IpCommandtriggerError
XSPI Ip Command Trigger Error

enumerator kStatus_XSPI_IllegalInstructionError
XSPI Illegal Instruction Error

enumerator kStatus_XSPI_SequenceExecutionTimeout
XSPI Sequence Execution Timeout

enumerator kStatus_XSPI_FradCheckError
XSPI Frad Check Error

enumerator kStatus_XSPI_TgQueueWritingError
XSPI Tg Queue Writing Error

enumerator kStatus_XSPI_IpsBusTransError
XSPI Ip Bus Transfer Error

enumerator kStatus_XSPI_AhbReadAccessAsserted
AHB read access is asserted.

enumerator kStatus_XSPI_AhbWriteAccessAsserted
AHB write access is asserted.

enumerator kStatus_XSPI_IPAccessAsserted
IP access is asserted.

enumerator kStatus_XSPI_AhbSubBufferFactorError
Wrong sub buffer factor input.

enumerator kStatus_XSPI_RegWriteLocked
Fail to write register due to write operation is locked.

enumerator kStatus_XSPI_WaterMarkIllegal
The watermark value to set is not in multiple of 4 bytes.

enumerator kStatus_XSPI_PageProgramWaitFlagAsserted
The page program wait flag already asserted.

enumerator kStatus_XSPI_IpAccessAddrSettingInvalid
The input address for Ip access is invalid.

enumerator kStatus_XSPI_RxBufferEntriesCountError
The count of entries of RX buffer is incorrect.

enumerator kStatus_XSPI_IpAccessIPCRInvalid
Access attributes to write IPCR are not correct.

enumerator kStatus_XSPI_IpAccessNotGranted
IP request not granted by arbiter.

enumerator kStatus_XSPI_IpReadFinished
Finish IP read all data stored in buffer.

enumerator kStatus_XSPI_IpWriteFinished
Finish IP write operation.

enumerator kStatus_XSPI_TxBufferUnderrun
XSPI attempted to pull data when the TX buffer is empty.

enumerator kStatus_XSPI_ArbiterGranted
XSPI arbiter granted.

enum _xspi_lut_instr

CMD definition of XSPI, use to form LUT instruction, xspi_lut_instr_t.

Values:

enumerator kXSPI_Command_STOP

Stop execution, de-assert CS.

enumerator kXSPI_Command_SDR

Transmit Command code to Flash, using SDR mode.

enumerator kXSPI_Command_RADDR_SDR

Transmit Row Address to Flash, using SDR mode.

enumerator kXSPI_Command_DUMMY_SDR

Leave data lines un-driven by xSPI controller, using SDR mode.

enumerator kXSPI_Command_MODE_SDR

Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kXSPI_Command_MODE2_SDR

Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kXSPI_Command_MODE4_SDR

Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kXSPI_Command_READ_SDR

Receive Read Data from Flash, using SDR mode.

enumerator kXSPI_Command_WRITE_SDR

Transmit Programming Data to Flash, using SDR mode.

enumerator kXSPI_Command_JMP_ON_CS

Stop execution, de-assert CS and save operand[7:0] as the instruction start pointer for next sequence

enumerator kXSPI_Command_RADDR_DDR

Transmit Row Address to Flash, using DDR mode.

enumerator kXSPI_Command_MODE_DDR

Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kXSPI_Command_MODE2_DDR

Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kXSPI_Command_MODE4_DDR

Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kXSPI_Command_READ_DDR

Receive Read Data from Flash, using DDR mode.

enumerator kXSPI_Command_WRITE_DDR

Transmit Programming Data to Flash, using DDR mode.

enumerator kXSPI_Command_LEARN_DDR

Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kXSPI_Command_DDR

Transmit Command code to Flash, using DDR mode.

enumerator kXSPI_Command_CADDR_SDR

Transmit Column Address to Flash, using SDR mode.

enumerator kXSPI_Command_CADDR_DDR
Transmit Column Address to Flash, using DDR mode.

enumerator kXSPI_Command_JUMP_TO_SEQ
Jump the LUT sequence pointed to by the operand.

enum _xspi_pad
pad definition of XSPI, use to form LUT instruction.

Values:

enumerator kXSPI_1PAD
Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kXSPI_2PAD
Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kXSPI_4PAD
Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kXSPI_8PAD
Transmit command/address and transmit/receive data only through DATA[7:0].

enum _xspi_flags
XSPI interrupt status flags.

Deprecated:
For Error flag please use xspi_error_flag_t as instead, for cmd execution and arbitration flag please use xspi_cmd_execution_arbitration_flag_t as instead.

Values:

enumerator kXSPI_DataLearningFailedFlag
Data learning failed.

enumerator kXSPI_DllAbortFlag
DLL Abort.

enumerator kXSPI_DllUnlockFlag
This field is set whenever DLL unlock event occurs, irrespective of flash memory access.

enumerator kXSPI_TxBufferFillFlag
If the field remains cleared, the TX buffer can be considered as full.

enumerator kXSPI_TxBufferUnderrunFlag
This field is set if the module tries to pull data when the TX buffer is empty

enumerator kXSPI_RxBufferOverflowFlag
This field is set when no more data can be pushed into the RX buffer from the serial flash memory device.

enumerator kXSPI_RxBufferDrainFlag
This field is set if SR[RXWE] is asserted.

enumerator kXSPI_AhbAbortErrorFlag
This flag can be set when AHB transaction error

enumerator kXSPI_AhbIllegalTransactionErrorFlag

enumerator kXSPI_AhbIllegalBurstSizeErrorFlag
The total burst size is more than prefetch size.

enumerator kXSPI_AhbBufferOverflowFlag
The size of the AHB access exceed the size of the AHB buffer.

enumerator kXSPI_AhbPerformanceMonitorOverflowFlag
AHB performance hit or miss counter overflow.

enumerator kXSPI_AhbReadAddressErrorFlag
The master send an AHB read address to that buffer is not within the address range of any sub-buffers or the address within multiple sub-buffers.

enumerator kXSPI_IllegalInstructionErrorFlag
This field is set when an illegal instruction is encountered by the controller in any of the sequences.

enumerator kXSPI_IpCmdtriggerErrorFlag
Write access to Rx buffer control reg when IP-triggered command is executing.

enumerator kXSPI_PageProgramWaitFlag
Indicates assertion of the page-program wait flag after writing to flash memory.

enumerator kXSPI_IpCommandExecutionDoneFlag
Indicates XSPI has completed a running IP command.

enumerator kXSPI_SequenceExecutionTimeoutFlag
Sequence execution timeout.

enumerator kXSPI_FradMatchErrorFlag

enumerator kXSPI_Frad0AccessErrorFlag

enumerator kXSPI_Frad1AccessErrorFlag

enumerator kXSPI_Frad2AccessErrorFlag

enumerator kXSPI_Frad3AccessErrorFlag

enumerator kXSPI_Frad4AccessErrorFlag

enumerator kXSPI_Frad5AccessErrorFlag

enumerator kXSPI_Frad6AccessErrorFlag

enumerator kXSPI_Frad7AccessErrorFlag

enumerator kXSPI_FradnAccErrorFlag
ORed value of all frad error flag,.

enumerator kXSPI_IpsErrorFlag

enumerator kXSPI_Tg0SfarErrorFlag

enumerator kXSPI_Tg1SfarErrorFlag

enumerator kXSPI_TgnIpcrErrorFlag

enumerator kXSPI_LockRegErrorFlag

enumerator kXSPI_ArbLockTimeoutFlag

enumerator kXSPI_ArbWinEventFlag

enum _xspi_error_flag
The enumeration of error flags. .

Values:

enumerator kXSPI_ErrorNoFradMatch

The transaction address does not lie within the address range of any FRAD.

enumerator kXSPI_ErrorFrad0Access

The transaction address lies within the address range of the FRAD0 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad1Access

The transaction address lies within the address range of the FRAD1 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad2Access

The transaction address lies within the address range of the FRAD2 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad3Access

The transaction address lies within the address range of the FRAD3 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad4Access

The transaction address lies within the address range of the FRAD4 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad5Access

The transaction address lies within the address range of the FRAD5 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad6Access

The transaction address lies within the address range of the FRAD6 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad7Access

The transaction address lies within the address range of the FRAD7 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorIpBusTransfer

A common error has occurred and XSPI has generated an IPS bus transfer error.

enumerator kXSPI_ErrorTg0Sfar

An SFAR write trigger an error while written to the TG0.

enumerator kXSPI_ErrorTg1Sfar

An SFAR write trigger an error while written to the TG1.

enumerator kXSPI_ErrorTg0Ipcr

An IPCR write trigger an error while written to the TG0.

enumerator kXSPI_ErrorTg1Ipcr

An IPCR write trigger an error while written to the TG1.

enumerator kXSPI_ErrorTimeout

A flash memory transaction has triggered a timeout error and has been terminated by SFP.

enumerator kXSPI_ArbitrationWin
The target group request was granted access.

enumerator kXSPI_ErrorArbitrationLock
Arbitration lock has timed out.

enumerator kXSPI_ErrorWriteLockedReg
An attempt to write a locked register.

enum _xspi_cmd_execution_arbitration_flag
The enumeration of cmd execution and arbitration. .

Values:

enumerator kXSPI_FlagIpCmdFinished
IP command transaction finished flag.

enumerator kXSPI_FlagAhbReadAddrError
Wrong AHB read address, XSPI return AHB error payload.

enumerator kXSPI_FlagAhbPerformanceMonitorOverflow
Buffer hit or miss counter overflow.

enumerator kXSPI_FlagIpCmdTriggerFail
Fail when set IP RX buffer watermark.

enumerator kXSPI_FlagPageProgramWait
Indicate assertion of the page-program wait flag after writing to flash memory.

enumerator kXSPI_FlagAhbBufferOverflow
The size of the AHB access exceeds the size of AHB buffer.

enumerator kXSPI_FlagAhbIllegalBurstSizeError
The total burst size of the AHB transacton is greater than the prefetch data size.

enumerator kXSPI_FlagAhbIllegalTransactionError
XSPI has not generated a response to the AHB bus due to an illegal transaction and expiration of the watchdog timer.

enumerator kXSPI_FlagAhbTransactionError
AHB transaction error.

enumerator kXSPI_FlagRxBufferWatermarkExceeded
The RX buffer watermark has been exceeded.

enumerator kXSPI_FlagRxBufferOverflow
The RX buffer cannot receive any more data from the SFM device.

enumerator kXSPI_FlagIllegalInstrError
The xspi encountered an illegal instruction in any sequence.

enumerator kXSPI_FlagDllUnlock
DLL unlock.

enumerator kXSPI_FlagTxBufferUnderRun
XSPI attempt to pull data when the TX buffer is empty.

enumerator kXSPI_FlagTxBufferFill
The TX buffer is filling.

enumerator kXSPI_FlagDllAbort
DLL terminate.

enumerator kXSPI_FlagDataLearningPatternFailure
The XSPI encountered a DATA_LEARN instruction in a sequence, but no sampling point is found for the data learning pattern.

enum _xspi_interrupt_enable

Values:

enumerator kXSPI_TransactionFinishIntEnable
Enable generation of an interrupt when an IP command transaction completes.

enumerator kXSPI_AhbReadAddrErrorIntEnable
Enable generation of an interrupt on occurrence of an AHB read address error.

enumerator kXSPI_AhbPerformanceMonitorOverflowIntEnable
Enable generation of an interrupt on a buffer hit or miss counter overflow.

enumerator kXSPI_IpCmdTrigFailErrorIntEnable
IP command trigger fail error interrupt enable.

enumerator kXSPI_PageProgramWaitIntEnable
Enable generation of an interrupt on page-program wait.

enumerator kXSPI_AhbBufferOverflowIntEnable
Enable generation of an interrupt on occurrence of an AHB buffer overflow.

enumerator kXSPI_AhbIllegalBurstSizeErrorIntEnable
Enable generation of an interrupt on occurrence of an AHB illegal burst size error.

enumerator kXSPI_AhbIllegalTransactionErrorIntEnable
Enable generation of an interrupt of an AHB illegal transaction error.

enumerator kXSPI_AhbAbortErrorIntEnable
Enable generation of an interrupt on occurrence of an AHB abort error.

enumerator kXSPI_RxBufferDrainIntEnable
Enable generation of an interrupt on occurrence of RX buffer drain.

enumerator kXSPI_RxBufferOverflowIntEnable
Enable generation of an interrupt on occurrence of RX buffer overflow.

enumerator kXSPI_IllegalInstrErrorIntEnable
Enable generation of an interrupt on occurrence of an illegal instruction error.

enumerator kXSPI_DllUnlockIntEnable
Enable generation of an interrupt on a DLL unlock event.

enumerator kXSPI_TxBufferUnderrunIntEnable
Enable generation of an interrupt on TX buffer underrun.

enumerator kXSPI_TxBufferFillIntEnable
Enable generation of an interrupt on TX buffer fill.

enumerator kXSPI_DataLearningPatternFailIntEnable
Enable generation of an interrupt on data learning pattern failure.

enumerator kXSPI_NoFradMatchErrorIntEnable
Enable the interrupt generated when the transaction address does not lie within the address range of any FRAD.

enumerator kXSPI_Frad0AccessErrorIntEnable
Enable the interrupt generated when the transaction address lies within the address range of the FRAD0 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad1AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD1 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad2AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD2 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad3AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD3 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad4AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD4 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad5AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD5 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad6AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD6 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad7AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD7 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_IpsErrIntEnable

Enable the interrupt generated when a common error has occurred an XSPI has generated an IPS bus transfer error.

enumerator kXSPI_Tg0SfarErrIntEnable

Enable the interrupt generated when an SFAR write triggers an error while being written to the target group 0.

enumerator kXSPI_Tg1SfarErrIntEnable

Enable the interrupt generated when an SFAR write triggers an error while being written to the target group 1.

enumerator kXSPI_Tg0IpcrErrIntEnable

Enable the interrupt generated when an IPCR write triggers an error while being written to the target group 0.

enumerator kXSPI_Tg1IpcrErrIntEnable

Enable the interrupt generated when an IPCR write triggers an error while being written to the target group 1.

enumerator kXSPI_TimeoutErrIntEnable

Enable the interrupt generated when a flash memory transaction triggers a timeout error and has been terminated by SFP.

enumerator kXSPI_ArbitrationWinIntEnable

Enable the interrupt generated when the target group queue request is granted access.

enumerator kXSPI_ArbitrationLockTimeoutErrIntEnable

Enable the interrupt generated when a flash memory transaction triggers a timeout error and has been terminated by SFP.

enumerator kXSPI_LockRegWriteErrIntEnable

Enable the interrupt generated by an attempt to write a locked register.

enum _xspi_external_signal

The enumeration of Idle signal drive IOFA[3] and IOFA[2].

Values:

enumerator XSPI_SignalIOFA2

Idle signal drive IOFA[2].

enumerator XSPI_SignalIOFA3

Idle signal drive IOFA[3].

enum _xspi_dll_mode

Dll Mode enumerations.

Values:

enumerator kXSPI_AutoUpdateMode

Dll mode : Auto update mode.

enumerator kXSPI_BypassMode

Dll mode : Bypass mode.

enum _xspi_sample_clk_source

The enumeration of sample clock source which is used to sample data from external devices.

Note: If external device support Data strobe signal, the sample clock should selected as kXSPI_SampleClkFromExternalDQS.

Values:

enumerator kXSPI_SampleClkFromNonInvertedFullySpeedDummyPadLoopback

Sample clock from non inverted fully speed dummy pad loopback.

enumerator kXSPI_SampleClkFromInvertedFullySpeedDummyPadLoopback

Sample clock from inverted fully speed dummy pad loopback.

enumerator kXSPI_SampleClkFromHalfSpeedDummyPadLoopback

Sample clock from half fully speed dummy pad loopback.

enumerator kXSPI_SampleClkFromDqsPadLoopback

Sample clock from Dqs pad loopback.

enumerator kXSPI_SampleClkFromExternalDQS

Sample clock from external DQS signal.

enum _xspi_data_learning_pad_select

The selection of data learning pad.

Values:

enumerator kXSPI_OnlyIO1

IO1 selected as pad to use for pattern matching IO.

enumerator kXSPI_BothIO1IO3
IO1 and IO3 selected as pad to use for pattern matching IO.

enum _xspi_device_addr_mode
External device's address mode.

Values:

enumerator kXSPI_DeviceByteAddressable
Byte addressable.

enumerator kXSPI_Device2ByteAddressable
Two bytes addressable.

enumerator kXSPI_Device4ByteAddressable
Four bytes addressable.

enum _xspi_device_interface_type
The enumeration of external device's interface type, hyper bus or SPI(including strandard and extended).

Note: Different interface result different settings.

Values:

enumerator kXSPI_StrandardExtendedSPI
External device follow SPI protocol, DQS_OUT_EN must be 0

enumerator kXSPI_HyperBus
External device follow hyper Bus protocol, DQS_OUT_EN must be 1

enum _xspi_hyper_bus_x16_mode
The enumeration of hyper bus X16 mode.

Values:

enumerator kXSPI_x16ModeDisable
X16 mode is disabled.

enumerator kXSPI_x16ModeEnabledOnlyData
X16 mode is enabled only for data.

enumerator kXSPI_x16ModeEnabledAddrCmdData
X16 mode is enabled for address, command, and data.

enum _xspi_ddr_data_aligned_clk
Enumeration defining the alignment modes of XSPI DDR data with respect to the internal reference clock.

Values:

enumerator kXSPI_DDRDataAlignedWithInternalRefClk
DDR data aligned with internal reference clock.

enumerator kXSPI_DDRDataAlignedWith2xInternalRefClk
DDR data aligned with 2X internal reference clock.

enum _xspi_command_type
Command type.

Values:

enumerator kXSPI_Command
XSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kXSPI_Config
XSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kXSPI_Read
XSPI operation: Read, only Rx Buffer is effective.

enumerator kXSPI_Write
XSPI operation: Read, only Tx Buffer is effective.

enum _xspi_mdad_mask_type
The enumeration of SFP MDAD mask type.

Values:

enumerator kXSPI_MdadMaskTypeAnd
Target master is mask & reference.

enumerator kXSPI_MdadMaskTypeOr
Target master is mask | reference.

enum _xspi_mdad_error_reason
The enumeration of MDAD error reasons.

Values:

enumerator kXSPI_ErrorOnlyDataSize
Error reason: Only Data size without required attributes.

enumerator kXSPI_ErrorOnlySeqId
Error reason: Only sequence ID without required attributes.

enumerator kXSPI_ErrorBothDataSizeSeqId
Error reason: Both Data size and sequence ID without required attributes.

enum _xspi_exclusive_access_lock_mode
Exclusive access lock.

Values:

enumerator kXSPI_ExclusiveAccessLockDisabled
Write permissions available for all masters based on their MDxACP evaluation.

enumerator kXSPI_ExclusiveAccessLockEnabled
Write permissions revoked for all domains, Any write transaction coming to this flash address will result in an error..

enumerator kXSPI_ExclusiveAccessLockExceptMasterId
Write permissions are revoked for all masters except the master ID specified by FRADn_WORD2[EALO] fields.

enum _xspi_secure_attribute
XSPI defines the secure attribute selection criteria for entry into descriptor queue...

Values:

enumerator kXSPI_AttributeMasterNonsecureOnly
Allow the bus attribute for this master to non-secure only

enumerator kXSPI_AttributeMasterSecureOnly
Allow the bus attribute for this master to secure only

enumerator kXSPI_AttributeMasterNonsecureSecureBoth
Allow the bus master's attribute: Both secure and non-secure

enum _xspi_descriptor_lock
Descriptor lock.

Values:

enumerator kXSPI_DescriptorLockDisabled
Descriptor registers can be written by any master

enumerator kXSPI_DescriptorLockEnabledTillHardReset
FRAD descriptor is read-only till next hard reset.

enumerator kXSPI_DescriptorLockEnabledExceptMasterId
Descriptors are read-only. MDnACP fields can be programmed only by the master with ID matching the MID.

enumerator kXSPI_DescriptorLockEnabled
Descriptor registers are read-only.

enum _xspi_target_group
The enumeration of target group.

Values:

enumerator kXSPI_TargetGroup0
Target groupe queue 0

enumerator kXSPI_TargetGroup1
Target groupe queue 1

enum _xspi_subbuffer_division_factor
Subbuffer division factor.

Values:

enumerator kXSPI_Division0
division factor 0

enumerator kXSPI_Division2
division factor 2 subbuffer size/2

enumerator kXSPI_Division4
division factor 4 subbuffer size/4

enumerator kXSPI_Division8
division factor 8 subbuffer size/8

enumerator kXSPI_Division16
division factor 16 subbuffer size/16

enum _xspi_ahb_sub_buffer_status
The enumeration of sub buffer status, empty, full, or partially full.

Values:

enumerator kXSPI_AhbSubBufferEmpty
Current sub buffer is empty.

enumerator kXSPI_AhbSubBufferFull
Current sub buffer is full.

enumerator kXSPI_AhbSubBufferPartiallyFull

Current sub buffer contains some entries.

enum _xspi_ppw_flag_clear_policy

The enumeration of XSPI page program waiting flag clear policy.

Values:

enumerator kXSPI_SoftwareClearPPWFlag

Application should clear the flag manually, after check the WIP of external flag is cleared.

enumerator kXSPI_HardwareClearPPWFlag

The flag is cleared by hardware automatically, the variable in type of xspi_device_status_reg_info_t should be defined.

enum _xspi_ahb_split_size

XSPI AHB Split Size These bits are used to control the split size when split function is enabled BFGENCR[SPLITEN] is '1'.

Values:

enumerator kXSPI_AhbSplitSize8b

Enable AHB transaction split and split size is 8 bytes.

enumerator kXSPI_AhbSplitSize16b

Enable AHB transaction split and split size is 16 bytes.

enumerator kXSPI_AhbSplitSize32b

Enable AHB transaction split and split size is 32 bytes.

enumerator kXSPI_AhbSplitSize64b

Enable AHB transaction split and split size is 64 bytes.

enumerator kXSPI_AhbSplitSizeDisabled

Disable AHB Transaction Split feature.

enum _xspi_ahb_alignment

XSPI AHB access towards flash is broken if this AHB alignment boundary is crossed. This field is only supported XSPI0 XSPI1.

Values:

enumerator kXSPI_AhbAlignmentNoLimit

no limit

enumerator kXSPI_AhbAlignment256BLimit

256B limit

enumerator kXSPI_AhbAlignment512BLimit

512B limit

enumerator kXSPI_AhbAlignment1KBLimit

1KB limit

enum _xspi_ahb_request_suspend_state

The enumeration of AHB request suspend state.

Values:

enumerator kXSPI_AhbRequestNoSuspended

None of AHB request is suspended.

enumerator kXSPI_AhbRequestSuspended
The AHB request is suspended.

enumerator kXSPI_AhbRequestNotApplicable
The AHB request is not applicable.

enumerator kXSPI_AhbReqestResumed
The AHB request is resumed.

enum _xspi_otfad_prefetch_boundary
The enumeration of OTFAD prefetch address boundary.

Values:

enumerator kXSPI_OTFADPrefetchNoBoundary
No prefetch address boundary.

enumerator kXSPI_OTFADPrefetchBoundary1K
Prefetch address boundary is 1K.

enum _xspi_byte_order
Byte ordering endianness.

Values:

enumerator kXSPI_64BitBE
64 bit big endian

enumerator kXSPI_32BitLE
32 bit little endian

enumerator kXSPI_32BitBE
32 bit big endian

enumerator kXSPI_64BitLE
64 bit little endian

enum _xspi_cache64_policy
The enumeration of cache64 policy.

Values:

enumerator kXSPI_Cache64PolicyNonCacheable
Non-cacheable.

enumerator kXSPI_Cache64PolicyWriteThrough
Write through.

enumerator kXSPI_Cache64PolicyWriteBack
Write back.

typedef enum *_xspi_lut_instr* xspi_lut_instr_t
CMD definition of XSPI, use to form LUT instruction, xspi_lut_instr_t.

typedef enum *_xspi_pad* xspi_pad_t
pad definition of XSPI, use to form LUT instruction.

typedef enum *_xspi_flags* xspi_flags_t
XSPI interrupt status flags.

Deprecated:
For Error flag please use xspi_error_flag_t as instead, for cmd execution and arbitration flag please use xspi_cmd_execution_arbitration_flag_t as instead.

typedef struct *_xspi_device_status_reg_info* xspi_device_status_reg_info_t

The structure of external device's status register information(location, priority).

Note: Only useful when AHB write to flash device is enabled, and use hardware to clear ppw flag.

typedef enum *_xspi_dll_mode* xspi_dll_mode_t

Dll Mode enumerations.

typedef union *_xspi_dll_para* xspi_dll_para_t

The union of user custommed DLL parameters, in case of kXSPI_BypassMode bypassModePara is used, in case of kXSPI_AutoUpdateMode autoUpdateModoPara is used.

typedef struct *_xspi_dll_config* xspi_dll_config_t

The structure of Delay-lock-loop configuration.

typedef enum *_xspi_sample_clk_source* xspi_sample_clk_source_t

The enumeration of sample clock source which is used to sample data from external devices.

Note: If external device support Data strobe signal, the sample clock should selected as kXSPI_SampleClkFromExternalDQS.

typedef enum *_xspi_data_learning_pad_select* xspi_data_learning_pad_select_t

The selection of data learning pad.

typedef struct *_xspi_data_learning_config* xspi_data_learning_config_t

The configuration of data learning.

Note: Data learning feature only supported in the DQS sampling method(kXSPI_SampleClkFromExternalDQS).

typedef struct *_xspi_sample_clk_config* xspi_sample_clk_config_t

Sample clock configuration, which used to sample data from external device.

typedef enum *_xspi_device_addr_mode* xspi_device_addr_mode_t

External device's address mode.

typedef enum *_xspi_device_interface_type* xspi_device_interface_type_t

The enumeration of external device's interface type, hyper bus or SPI(including strandard and extended).

Note: Different interface result different settings.

typedef enum *_xspi_hyper_bus_x16_mode* xspi_hyper_bus_x16_mode_t

The enumeration of hyper bus X16 mode.

typedef union *_xspi_device_interface_settings* xspi_device_interface_settings_t

The union of external device's settings, if external device following hyper bus protocol #hyperBusSettings take effects, if external device following SPI protocol #strandardExtendedSPISettings take effects.

typedef enum *_xspi_ddr_data_aligned_clk* xspi_ddr_data_aligned_clk_t

Enumeration defining the alignment modes of XSPI DDR data with respect to the internal reference clock.

typedef struct *_xspi_device_ddr_config* xspi_device_ddr_config_t
: The structure of device DDR configuration.

typedef struct *_xspi_device_config* xspi_device_config_t
: External device configuration items.

typedef enum *_xspi_command_type* xspi_command_type_t
: Command type.

typedef enum *_xspi_mdad_mask_type* xspi_mdad_mask_type_t
: The enumeration of SFP MDAD mask type.

typedef enum *_xspi_mdad_error_reason* xspi_mdad_error_reason_t
: The enumeration of MDAD error reasons.

typedef enum *_xspi_exclusive_access_lock_mode* xspi_exclusive_access_lock_mode_t
: Exclusive access lock.

typedef enum *_xspi_secure_attribute* xspi_secure_attribute_t
: XSPI defines the secure attribute selection criteria for entry into descriptor queue...

typedef struct *_xspi_mdad_config* xspi_mdad_config_t
: MDAD configuration.

typedef struct *_xspi_sfp_mdad_config* xspi_sfp_mdad_config_t
: The structure of SFP MDAD configurations for all target groups.

typedef enum *_xspi_descriptor_lock* xspi_descriptor_lock_t
: Descriptor lock.

typedef struct *_xspi_frad_config_t* xspi_frad_config_t
: FRAD configuration.

typedef struct *_xspi_sfp_frad_config* xspi_sfp_frad_config_t
: The structure of SFP FRAD configurations.

typedef struct *_xspi_frad_transaction_info* xspi_frad_transaction_info_t
: The structure of latest FRAD transaction information.

typedef struct *_xspi_ip_access_config* xspi_ip_access_config_t
: Configurations of IP access(including IP read and write).

typedef enum *_xspi_target_group* xspi_target_group_t
: The enumeration of target group.

typedef struct *_xspi_tg_add_write_status* xspi_tg_add_write_status_t
: The status of latest target group address write operation.

typedef struct *_xspi_transfer* xspi_transfer_t
: Transfer structure used for XSPI functional interface.

typedef struct *_xspi_handle* xspi_handle_t

typedef void (*xspi_transfer_callback_t)(XSPI_Type *base, *xspi_handle_t* *handle, *status_t* status, void *userData)
: XSPI transfer callback function.

typedef enum *_xspi_subbuffer_division_factor* xspi_subbuffer_division_factor_t
: Subbuffer division factor.

typedef struct *_xspi_ahbBuffer_sub_buffer_config* xspi_ahbBuffer_sub_buffer_config_t
: The structure of sub-buffer configurations.

typedef struct *_xspi_ahbBuffer_config* xspi_ahbBuffer_config_t

Structure of AHB buffer configurations.

Note: When an AHB read access comes it is assigned to a buffer based on its master ID. Then it further assigned to a sub-buffer based on which sub-buffer address range this transaction lies into.

Note: When sub-buffer division is hit, in case of a buffer miss the controller will fetch the data equal to the size of that sub-buffer.

Note: If sub-buffer division is enabled for a buffer the hit/miss check is done at sub-buffer level. And if prefetch is enable, the prefetch takes place for individual sub-buffers.

---

typedef enum *_xspi_ahb_sub_buffer_status* xspi_ahb_sub_buffer_status_t

The enumeration of sub buffer status, empty, full, or partially full.

typedef struct *_xspi_ahbBuffer_perf_monitor_result* xspi_ahbBuffer_perf_monitor_result_t

The result of AHB buffer performance monitor, including buffer miss count and buffer hit count.

typedef enum *_xspi_ppw_flag_clear_policy* xspi_ppw_flag_clear_policy_t

The enumeration of XSPI page program waiting flag clear policy.

typedef struct *_xspi_ahb_write_config* xspi_ahb_write_config_t

The configuration of AHB write access.

typedef enum *_xspi_ahb_split_size* xspi_ahb_split_size_t

XSPI AHB Split Size These bits are used to control the split size when split function is enabled BFGENCR[SPLITEN] is '1'.

typedef enum *_xspi_ahb_alignment* xspi_ahb_alignment_t

XSPI AHB access towards flash is broken if this AHB alignment boundary is crossed. This field is only supported XSPI0 XSPI1.

typedef struct *_xspi_ahb_access_config* xspi_ahb_access_config_t

The structure of AHB access configurations, including AHB buffer settings, AHB Read Seq ID, AHB write settings.

typedef struct *_xspi_ahb_read_error_info* xspi_ahb_read_error_info_t

The structure of information when AHB read error occur.

typedef enum *_xspi_ahb_request_suspend_state* xspi_ahb_request_suspend_state_t

The enumeration of AHB request suspend state.

typedef struct *_xspi_ahb_request_suspend_info* xspi_ahb_request_suspend_info_t

The structure of information when AHB reqest is suspended.

typedef enum *_xspi_otfad_prefetch_boundary* xspi_otfad_prefetch_boundary_t

The enumeration of OTFAD prefetch address boundary.

typedef enum *_xspi_byte_order* xspi_byte_order_t

Byte ordering endianness.

typedef struct *_xspi_config* xspi_config_t

XSPI configuration structure.

typedef enum *_xspi_cache64_policy* xspi_cache64_policy_t

The enumeration of cache64 policy.

typedef struct *_xspi_cache64_config* xspi_cache64_region_config_t

The structure of cache64 configuration.

Region0 range from base address to region0TopAddr[28:10] + 0x3FF, region1 range from region0TopAddr[28:10] + 0x400 to region1TopAddr[28:10] + 0x3FF, region1TopAddr[28:10] + 0x400 to end address.

FSL_XSPI_DRIVER_VERSION

XSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

Formula to form XSPI instructions in LUT table.

XSPI_TARGET_GROUP_COUNT

XSPI_SFP_FRAD_COUNT

XSPI_IP_RX_BUFFER_SIZE

XSPI_IP_TX_BUFFER_SIZE

uint32_t XSPI_GetInstance(XSPI_Type *base)

Get the instance number for XSPI.

Parameters

- base – XSPI base pointer.

status_t XSPI_CheckAndClearError(XSPI_Type *base, uint32_t status)

Check and clear IP command execution errors.

Parameters

- base – XSPI base pointer.
- status – interrupt status.

void XSPI_Init(XSPI_Type *base, const *xspi_config_t* *ptrConfig)

Initializes the XSPI module and internal state.

This function configures the XSPI with the input configure parameters. Users should call this function before any XSPI operations.

Parameters

- base – XSPI peripheral base address.
- ptrConfig – XSPI configure structure.

void XSPI_GetDefaultConfig(*xspi_config_t* *ptrConfig)

Gets default settings for XSPI.

Warning: The parameter **devconfig** will be removed in next release.

Parameters

- base – XSPI peripheral base address.
- ptrConfig – XSPI configuration structure.

void XSPI_Deinit(XSPI_Type *base)

Deinitialize the XSPI module.

Clears the XSPI state and XSPI module registers.

Parameters

- base – XSPI peripheral base address.
- devconfig – Flash configuration parameters.

void XSPI_UpdateLUT(XSPI_Type *base, uint8_t index, const uint32_t *cmd, uint8_t count)

Updates the LUT table.

Parameters

- base – XSPI peripheral base address.
- index – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 10 instructions and occupy 5*32-bit memory.
- cmd – Command sequence array.
- count – Number of instruction-operand pairs

static inline void XSPI_SetOTFADPrefetchBoundary(XSPI_Type *base, *xspi_otfad_prefetch_boundary_t* boundary)

Set OTFAD prefetch boundary.

Parameters

- base – XSPI peripheral base address.
- boundary – Prefetch boundary to set, in type of xspi_otfad_prefetch_boundary_t.

static inline void XSPI_UpdateByteOrder(XSPI_Type *base, *xspi_byte_order_t* byteOrder)

static inline void XSPI_EnableModule(XSPI_Type *base, bool enable)

Enable or disable the XSPI module.

Parameters

- base – XSPI peripheral base address.
- enable – true means enable XSPI, false means disable.

static inline bool XSPI_CheckModuleEnabled(XSPI_Type *base)

Check if the XSPI module is enabled.

Parameters

- base – XSPI peripheral base address.

Return values

- true – XSPI module is enabled.
- false – XSPI module is disabled.

void XSPI_ResetSfmAndAhbDomain(XSPI_Type *base)

Reset Serial flash memory domain and AHB domain at the same time.

Note: Resetting only the Serial flash memory domain and AHB domain may cause undesirable side effects.

Parameters

- base – XSPI peripheral base address.

static inline void XSPI_ResetTgQueue(XSPI_Type *base)

Reset IPS target group queue.

Parameters

- base – XSPI peripheral base address.

static inline void XSPI_SoftwareReset(XSPI_Type *base)

Software reset flash memory domain, AHB domain, and Target group at the same time.

Parameters

- base – XSPI peripheral base address.

static inline bool XSPI_CheckGlobalConfigLocked(XSPI_Type *base)

Check if write access to registers(MGC, MRC, MTO, SFP_ARB_TIMEOUT, SFP_LUT_ENn, LUTn, BFGENCR) is locked.

Parameters

- base – XSPI peripheral base address.

Return values

- false – Write access to listed registers is not locked.
- true – Write access to listed registers is locked.

void XSPI_SetHyperBusX16Mode(XSPI_Type *base, *xspi_hyper_bus_x16_mode_t* x16Mode)

Set Hyper bus X16 mode.

Parameters

- base – XSPI peripheral base address.
- x16Mode – Specify X16 mode.

status_t XSPI_UpdateDeviceAddrMode(XSPI_Type *base, *xspi_device_addr_mode_t* addrMode)

Update address mode to access external device.

Parameters

- base – XSPI peripheral base address.
- addrMode – Specify the address mode to update.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail to update address mode due to AHB read access asserted.
- kStatus_XSPI_IPAccessAsserted – Fail to update address mode due to a IP access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail to update address mode due to AHB write access asserted.
- kStatus_Success – Successfully to update address mode.

static inline void XSPI_EnableVariableLatency(XSPI_Type *base, bool enable)

Enable/disable variable latency for XSPI.

Note: Only useful for hyper bus device.

Note: It is application responsibility to enable variable latency for external hyper bus device(by change device's register).

Parameters

- base – XSPI peripheral base address.
- enable – Specify operation:
  - **true** Enable variable latency;
  - **false** Disable variable latency.

static inline void XSPI_EnableDozeMode(XSPI_Type *base, bool enable)

Enable/disable Doze mode for XSPI controller.

Note: Once doze mode is enabled, the XSPI controller will enter disable mode when system in low power modes.

Parameters

- base – XSPI peripheral base address
- enable – Specify operation:
  - **true** Enable Doze mode;
  - **false** Disable Doze mode.

static inline void XSPI_SetSignalOutputValue(XSPI_Type *base, uint32_t signalMask, bool outputLogic)

Specify the logic level of the XSPI IOFA[3] and IOFA[2] output in the inactive state.

Parameters

- base – XSPI peripheral base address
- signalMask – Should be the OR'ed value of _xspi_external_signal.
- outputLogic – Output logic of selected signals,
  - **false** Output logic 0;
  - **true** Output logic 1.

static inline void XSPI_EnableInvertedSerialClockOutput(XSPI_Type *base, bool enable)

Enable/disable inverted serial clock output for XSPI controller.

Note: It is application responsibility to enable CLKn for external device(by change device's register).

Parameters

- base – **[in]** XSPI peripheral base address
- enable – **[in]** Enable/disable inverted serial clock output:
  - **true** Enable inverted serial clock output;
  - **false** Disable inverted serial clock output.

void XSPI_UpdateDllValue(XSPI_Type *base, *xspi_dll_config_t* *ptrDllConfig, bool enableDDR, bool enableX16Mode, uint32_t xspiRootClk)

Update DLL configuration.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrDllConfig – **[in]** Pointer to the DLL configuration.
- enableDDR – **[in]** DDR mode is enabled or not.
- enableX16Mode – **[in]** X16 mode is enabled or not.
- xspiRootClk – **[in]** The frequency of xspi root clock, the unit is Hz.

status_t XSPI_SetDataLearningConfig(XSPI_Type *base, *xspi_data_learning_config_t* *ptrDataLearningConfig)

Set Data learning configurations.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrDataLearningConfig – **[in]** Pointer to data learning configuration.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail to set data learning configuration due to AHB read access asserted.
- kStatus_XSPI_IPAccessAsserted – Fail to set data learning configuration due to a IP access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail to set data learning configuration due to AHB write access asserted.
- kStatus_Success – Successfully to set Data learning configurations.

static inline bool XSPI_CheckDataLearningFailure(XSPI_Type *base)

Check if data learning failure has detected.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Data learning has failed.
- false – Data learning not fail.

static inline void XSPI_GetDataLearningEdgeMatchSignature(XSPI_Type *base, uint8_t *posEdgeMatch, uint8_t *negEdgeMatch)

Get data learning positive and negative edge match signature.

Parameters

- base – **[in]** XSPI peripheral base address.
- posEdgeMatch – **[out]** Pointer to the memory to store positive edge match signature.
- negEdgeMatch – **[out]** Pointer to the memory to store negative edge match signature.

status_t XSPI_SetDeviceConfig(XSPI_Type *base, *xspi_device_config_t* *devConfig)

Set device configuration.

Parameters

- base – **[in]** XSPI peripheral base address.
- devConfig – **[in]** Pointer to device configuration.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail to set device configuration due to AHB read access asserted.
- kStatus_XSPI_IPAccessAsserted – Fail to set device configuration due to a IP access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail to set device configuration due to AHB write access asserted.
- kStatus_Success – Successfully to set device configurations.

static inline void XSPI_EnableInterrupts(XSPI_Type *base, uint64_t mask)

Enables the XSPI interrupts.

Parameters

- base – XSPI peripheral base address.
- mask – XSPI interrupt source, should be the OR'ed value of xspi_interrupt_enable_t.

static inline void XSPI_DisableInterrupts(XSPI_Type *base, uint64_t mask)

Disables the XSPI interrupts.

Parameters

- base – XSPI peripheral base address.
- mask – XSPI interrupt source, should be the OR'ed value of xspi_interrupt_enable_t.

static inline void XSPI_EnableTxDMA(XSPI_Type *base, bool enable)

Enables or disables XSPI IP Tx FIFO DMA requests.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void XSPI_EnableRxDMA(XSPI_Type *base, bool enable)

Enables or disables XSPI IP Rx FIFO DMA requests.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t XSPI_GetTxFifoAddress(XSPI_Type *base)

Gets XSPI IP tx fifo address for DMA transfer.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The tx fifo address.

static inline uint32_t XSPI_GetRxFifoAddress(XSPI_Type *base)

Gets XSPI IP rx fifo address for DMA transfer.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The rx fifo address.

static inline uint32_t XSPI_GetErrorStatusFlags(XSPI_Type *base)

Get error status flags.

Parameters

- base – SPC peripheral base address.

Returns

All asserted error status flags, should be the OR'ed value of XSPI_GetErrorStatusFlags.

static inline void XSPI_ClearErrorStatusFlags(XSPI_Type *base, uint32_t flags)

Clear input error status flags.

Parameters

- base – SPC peripheral base address.
- flags – Error flags to clear, the OR'ed value of XSPI_GetErrorStatusFlags.

static inline uint32_t XSPI_GetInterruptStatusFlags(XSPI_Type *base)

Get the XSPI interrupt status flags.

Deprecated:

Please use XSPI_GetErrorStatusFlags() as instead.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

interrupt status flag, use status flag to AND xspi_flags_t could get the related status.

static inline bool XSPI_CheckIpRequestGranted(XSPI_Type *base)

Check if the IP access request was granted access if enable corresponding interrupt source, the XSPI IRQ will be asserted.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Ip request granted.
- false – Ip reqest not granted.

static inline bool XSPI_GetBusIdleStatus(XSPI_Type *base)

Return whether the bus is idle.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Bus is idle.

- false – Bus is busy.

static inline bool XSPI_CheckAhbReadAccessAsserted(XSPI_Type *base)

Check if AHB read access has been requested or is ongoing.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – AHB read access is requested or is ongoing.
- false – AHB read access is not requested and is not ongoing.

static inline bool XSPI_CheckAhbWriteAccessAsserted(XSPI_Type *base)

Check if AHB write access has been requested or is ongoing.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – AHB read access is requested or is ongoing.
- false – AHB read access is not requested and is not ongoing.

static inline uint32_t XSPI_GetCmdExecutionArbitrationStatusFlags(XSPI_Type *base)

Get asserted flags about SFM command execution and arbitration.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The assert flags about SFM command execution and arbitration, should be the OR'ed value of xspi_cmd_execution_arbitration_flag_t.

static inline void XSPI_ClearCmdExecutionArbitrationStatusFlags(XSPI_Type *base, uint32_t flags)

Clear asserted flags about SFM command execution and arbitration.

Parameters

- base – **[in]** XSPI peripheral base address.
- flags – **[in]** The mask of flags to clear, should be the OR'ed value of xspi_cmd_execution_arbitration_flag_t.

static inline void XSPI_ResetTxRxBuffer(XSPI_Type *base, bool txFifo, bool rxFifo)

Clear the XSPI IP TX/RX buffer logic.

Parameters

- base – **[in]** XSPI peripheral base address.
- txFifo – **[in]** Pass true to reset TX FIFO.
- rxFifo – **[in]** Pass true to reset RX FIFO.

static inline void XSPI_ClearTxBuffer(XSPI_Type *base)

Clear TX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

status_t XSPI_UpdateTxBufferWaterMark(XSPI_Type *base, uint32_t waterMark)

Update watermark for TX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- waterMark – **[in]** The watermark to set, the unit is byte, should be the multiple of 4 byte.

Return values

- kStatus_XSPI_IPAccessAsserted – Fail to update watermark for Tx buffer, due to IP access is asserted.
- kStatus_XSPI_WaterMarkIllegal – Fail to update watermark for Tx buffer, due to input watermark is not the multiple of 4 bytes.
- kStatus_Success – Successful to update watermark.

static inline bool XSPI_CheckTxBuffLockOpen(XSPI_Type *base)

Check if IP manager can write to TX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Tx buffer lock is open.
- false – Tx buffer lock is not open.

static inline void XSPI_WriteTxBuffer(XSPI_Type *base, uint32_t data)

Writes data into IPS TX Buffer.

Parameters

- base – **[in]** XSPI peripheral base address
- data – **[in]** The data bytes to send.

static inline void XSPI_ClearRxBuffer(XSPI_Type *base)

Clear RX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

static inline uint32_t XSPI_ReadRxBuffer(XSPI_Type *base, uint8_t fifoIndex)

Receive data from IPX RX FIFO.

Parameters

- base – **[in]** XSPI peripheral base address
- fifoIndex – **[in]** Source fifo index.

Returns

The data in the FIFO.

static inline void XSPI_TriggerRxBufferPopEvent(XSPI_Type *base)

Trigger a pop event for RX buffer pop event.

Note: Each pop event discard watermark + 1 enties from RX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

status_t XSPI_UpdateRxBufferWaterMark(XSPI_Type *base, uint32_t waterMark)

Update watermark for RX buffer.

```
Set watermark as 4 bytes:
XSPI_UpdateRxBufferWaterMark(XSPI0, 4UL);
Set watermark as 8 bytes:
XSPI_UpdateRxBufferWaterMark(XSPI0, 8UL);
```

Parameters

- base – **[in]** XSPI peripheral base address.
- waterMark – **[in]** Specify the number of bytes in the RX buffer which causes XSPI to assert the watermark exceeded flag, should be in multiple of 4 bytes.

Return values

- kStatus_XSPI_IPAccessAsserted – Fail to update watermark for Rx buffer, due to IP access is asserted.
- kStatus_XSPI_WaterMarkIllegal – Fail to update watermark for Tx buffer, due to input watermark is not the multiple of 4 bytes.
- kStatus_Success – Successful to update watermark.

static inline bool XSPI_CheckRxBufferWaterMarkExceed(XSPI_Type *base)

Check if RX buffer watermark is exceed.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – The RX buffer watermark has been excceded.
- false – The RX buffer watermark has not been exceeded.

static inline uint32_t XSPI_GetRxBufferAvailableBytesCount(XSPI_Type *base)

Get RX buffer aviailable bytes count.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The available counts if bytes in RX buffer.

static inline uint32_t XSPI_GetRxBufferRemovedBytesCount(XSPI_Type *base)

Get counts of bytes already removed from RX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

Counts of removed bytes.

void XSPI_SetSFPFradEALMode(XSPI_Type *base, *xspi_exclusive_access_lock_mode_t* ealMode, uint8_t fradId)

Set exclusive access lock mode for the specific frad..

Parameters

- base – **[in]** XSPI peripheral base address.
- ealMode – **[in]** Specify the exclusive access lock mode.

- fradId – **[in]** Specify the frad.

void XSPI_UpdateSFPConfig(XSPI_Type *base, *xspi_sfp_mdad_config_t* *ptrSfpMdadConfig, *xspi_sfp_frad_config_t* *ptrSfpFradConfig)

Update SFP configurations, including MDAD configurations and FRAD configurations.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrMdadConfig – **[in]** Pointer to the SFP MDAD configuration.
- ptrFradConfig – **[in]** Pointer to the SFP FRAD configuration.

static inline bool XSPI_CheckSFPFradEnabled(XSPI_Type *base)

Check if SFP FRAD check is enabled for IP write access.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- false – SFP FRAD check is disabled.
- true – SFP FRAD check is enabled for IP write access.

xspi_mdad_error_reason_t XSPI_GetMdadErrorReason(XSPI_Type *base, *xspi_target_group_t* tgId)

Get MDAD check error reason for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.

Returns

The details of MDAD error reason, in type of xspi_mdad_error_reason_t.

void XSPI_GetFradLastTransactionsInfo(XSPI_Type *base, *xspi_frad_transaction_info_t* *ptrInfo, uint8_t fradId)

Get FARD latest transaction information.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrInfo – **[out]** Pointer to the variable in type of xspi_frad_transaction_info_t to store information.
- fradId – **[in]** Specify the frad Id.

status_t XSPI_UpdateSFPArbitrationLockTimeoutCounter(XSPI_Type *base, uint32_t countValue)

Update SFP arbitration lock timeout counter.

Note: The SFP arbitration lock time out counter starts when Page program wait flag asserted.

Parameters

- base – **[in]** XSPI peripheral base address.
- countValue – **[in]** The count value, specify the time in IPS clock cycles.

Return values

- kStatus_XSPI_RegWriteLocked – Write operation to related register is locked.
- kStatus_Success – Success to update timeout counter.

void XSPI_ClearTgAddrWriteStatus(XSPI_Type *base, *xspi_target_group_t* tgId)

Clear address write status for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group to clear address write status.

void XSPI_GetTgAddrWriteStatus(XSPI_Type *base, *xspi_target_group_t* tgId, *xspi_tg_add_write_status_t* *ptrStatus)

Get address write status for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.
- ptrStatus – **[out]** Pointer to the variable in type of xspi_tg_add_write_status_t to store address write status.

void XSPI_UnlockIpAccessArbitration(XSPI_Type *base, *xspi_target_group_t* tgId)

Unlock Ip access arbitration.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.

static inline bool XSPI_CheckIPAccessAsserted(XSPI_Type *base)

Check if IP access is asserted.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- false – The Access triggered by IP bus is not asserted.
- true – The Access triggered by IP bus is asserted.

static inline void XSPI_ClearIPAccessSeqPointer(XSPI_Type *base)

Clear Ip access sequence pointer.

Parameters

- base – **[in]** XSPI peripheral base address.

status_t XSPI_UpdateIPAccessTimeoutCounter(XSPI_Type *base, uint32_t countValue)

Update the count of SFP access timeout counter.

Note: The counter starts when any IP access pass SFP check(if enabled), and request is granted by XSPI arbiter and XSPI is IDLE.

Note: The counter does not start in case of IP access was granted by XSPI is not IDLE.

Parameters

- base – XSPI peripheral base address.
- countValue – The count value, specify the time in IPS clock cycles.

Return values

- kStatus_XSPI_RegWriteLocked – Write operation to related register is locked.
- kStatus_Success – Success to update timeout counter.

static inline bool XSPI_CheckIPAccessGranted(XSPI_Type *base)

Check if IP access is granted by XSPI arbitration.

Deprecated:

Use XSPI_CheckIpRequestGranted() as instead

Parameters

- base – XSPI peripheral base address.

Return values

- true – The IP access is granted arbitration.
- false – No IP access is queued.

static inline bool XSPI_CheckIpWriteTriggered(XSPI_Type *base)

Check if IP write access is triggered.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – The IP write access is granted arbitration.
- false – No IP write access is queued.

static inline bool XSPI_CheckIpReadTriggered(XSPI_Type *base)

Check if IP read access is triggered.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – The IP read access is granted arbitration.
- false – No IP read access is queued.

static inline bool XSPI_CheckFSMValid(XSPI_Type *base)

Check if IPS transfer is granted arbitration or execution.

Note: The FSMSTAT[VLD] goes to 0 once the IPS transfer is completed and XSPI is IDLE.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Valid, the IPS transfer is granted arbitration or execution.
- false – Not valid, no IPS transfer is queued.

status_t XSPI_StartIpAccess(XSPI_Type *base, uint32_t addr, uint8_t seqIndex, size_t byteSize, *xspi_target_group_t* tgId, bool lockArbitration)

Start IP access(including read and write) in blocking way.

Parameters

- base – **[in]** XSPI peripheral base address.
- addr – **[in]** Address of external device to read/write.
- seqIndex – **[in]** Sequence Id of the pre-programmed LUT.
- byteSize – **[in]** Size of data to read/write, the unit of byte.
- tgId – **[in]** Specify the target group used to write/read.
- lockArbitration – **[in]** Lock arbitration or not.

Return values

- kStatus_XSPI_IpAccessAddrSettingInvalid – Wrong Address input.
- kStatus_XSPI_IpAccessIPCRInvalid – Wrong seqindex or bytesize input.
- kStatus_Success – Success to start Ip access.

status_t XSPI_StartIpAccessNonBlocking(XSPI_Type *base, uint32_t addr, uint8_t seqIndex, size_t byteSize, *xspi_target_group_t* tgId, bool lockArbitration)

Start IP access in non-blocking way.

Parameters

- base – **[in]** XSPI peripheral base address.
- addr – **[in]** Address of external device to read/write.
- seqIndex – **[in]** Sequence Id of the pre-programmed LUT.
- byteSize – **[in]** Size of data to transfer, the unit of byte.
- tgId – **[in]** Specify the target group to use.
- lockArbitration – **[in]** Lock arbitration or not.

Return values

- kStatus_Success – Success to start Ip access.
- kStatus_XSPI_IpAccessIPCRInvalid – Invalid to set IPCR register.
- kStatus_XSPI_IpAccessAddrSettingInvalid – Invalid to set SFAR register.

status_t XSPI_SetIpAccessConfig(XSPI_Type *base, *xspi_ip_access_config_t* *ptrIpAccessConfig)

Set IP access configurations, including SFP configurations, sfp arbitration lock timeout value, Ip access timeout value.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrIpAccessConfig – **[in]** Pointer to the variable which contains Ip access configurations.

Return values

- kStatus_XSPI_RegWriteLocked – Write operation to related register is locked.
- kStatus_Success – Success to update timeout counter.

status_t XSPI_WriteBlocking(XSPI_Type *base, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – **[in]** XSPI peripheral base address
- buffer – **[in]** Pointer to the buffer to send.
- size – **[in]** The number of data bytes to send

Return values

- kStatus_Success – write success without error
- kStatus_XSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_XSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_XSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t XSPI_ReadBlocking(XSPI_Type *base, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – **[in]** XSPI peripheral base address
- buffer – **[out]** Pointer to the buffer to store read data.
- size – **[in]** The number of data bytes to receive

Return values

- kStatus_Success – read success without error
- kStatus_XSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_XSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_XSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t XSPI_TransferBlocking(XSPI_Type *base, *xspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Note: In case of flash page program, if WEL(Write enable) automatically cleared after program operation, then WEL should be asserted before invoking this API and the transfer size should not bigger than page size.

Parameters

- base – **[in]** XSPI peripheral base address
- xfer – **[in]** pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error
- kStatus_XSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_XSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_XSPI_IpCommandGrantTimeout – IP command grant timeout detected

void XSPI_TransferCreateHandle(XSPI_Type *base, *xspi_handle_t* *handle, *xspi_transfer_callback_t* callback, void *userData)

Initialize the XSPI handle which is used in transactional functions.

Parameters

- base – **[in]** XSPI peripheral base address.
- handle – **[in]** pointer to xspi_handle_t structure to store the transfer state.
- callback – **[in]** pointer to user callback function.
- userData – **[in]** user parameter passed to the callback function.

status_t XSPI_TransferNonBlocking(XSPI_Type *base, *xspi_handle_t* *handle, *xspi_transfer_t* *xfer)

Perform a interrupt non-blocking transfer on the XSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call XSPI_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_XSPI_Busy, the transfer is finished. For XSPI_Read, the dataSize should be multiple of rx watermark level, or XSPI could not read data properly.

Parameters

- base – **[in]** XSPI peripheral base address.
- handle – **[in]** pointer to xspi_handle_t structure which stores the transfer state.
- xfer – **[in]** pointer to xspi_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_XSPI_Busy – Previous transmission still not finished.

status_t XSPI_TransferGetCount(XSPI_Type *base, *xspi_handle_t* *handle, size_t *count)

Get the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – **[in]** XSPI peripheral base address.
- handle – **[in]** pointer to xspi_handle_t structure which stores the transfer state.
- count – **[out]** 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 XSPI_TransferAbort(XSPI_Type *base, *xspi_handle_t* *handle)

Abort 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 – **[in]** XSPI peripheral base address.
- handle – **[in]** pointer to xspi_handle_t structure which stores the transfer state

static inline void XSPI_ClearAhbBuffer(XSPI_Type *base)

Blocks to clear AHB buffer and abort ongoing prefetch transaction(if prefetch is enabled.)

Note: The function return only after all AHB buffer pointers are cleared.

Parameters

- base – **[in]** XSPI peripheral base address.

status_t XSPI_EnableAhbBufferWriteFlush(XSPI_Type *base, bool enable)

Enable/disable the clearing of AHB read prefetch buffers when the same flash address is written by an AHB or IP command.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Used to enable/disable write flush.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted.
- kStatus_Success – Successfully to enable/disable AHB buffer write flush.

status_t XSPI_SetAhbBufferConfig(XSPI_Type *base, *xspi_ahbBuffer_config_t* *ptrBuffer0Config,
xspi_ahbBuffer_config_t *ptrBuffer1Config,
xspi_ahbBuffer_config_t *ptrBuffer2Config,
xspi_ahbBuffer_config_t *ptrBuffer3Config)

Set AHB buffer configurations.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrBuffer0Config – **[in]** Pointer to the variable which contain buffer0 configurations.
- ptrBuffer1Config – **[in]** Pointer to the variable which contain buffer1 configurations.
- ptrBuffer2Config – **[in]** Pointer to the variable which contain buffer2 configurations.
- ptrBuffer3Config – **[in]** Pointer to the variable which contain buffer3 configurations.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted.
- kStatus_Success – Success to set AHB buffer configurations.

status_t XSPI_UpdateAhbBufferSize(XSPI_Type *base, uint16_t buf0Size, uint16_t buf1Size, uint16_t buf2Size, uint16_t buf3Size)

Set Buffer size for all 4 AHB buffers.

Parameters

- base – **[in]** XSPI peripheral base address.
- buf0Size – **[in]** Specify size of AHB buffer0, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.
- buf1Size – **[in]** Specify size of AHB buffer1, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.
- buf2Size – **[in]** Specify size of AHB buffer2, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.
- buf3Size – **[in]** Specify size of AHB buffer3, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted.
- kStatus_Success – Success to set AHB buffer size.

xspi_ahb_sub_buffer_status_t XSPI_GetAhbSubBufferStatus(XSPI_Type *base, uint8_t ahbBufferId, uint8_t subBufferId)

Get status of AHB sub buffer.

Note: This function only useful when sub-division feature of the selected AHB buffer is enabled.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** The Id of AHB buffer, range from 0 to 3.
- subBufferId – **[in]** The Id of AHB buffer sub division, range from 0 to 3.

Returns

Current status of selected AHB sub buffer, in type of xspi_ahb_sub_buffer_status_t.

static inline void XSPI_StartAhbBufferPerfMonitor(XSPI_Type *base)

Start AHB buffer performance monitor.

Parameters

- base – **[in]** XSPI peripheral base address.

void XSPI_EnableAhbBufferPerfMonitor(XSPI_Type *base, uint8_t ahbBufferId, uint8_t subBufferId)

Enable AHB buffer performance monitor for selected AHB buffer's sub buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the selected AHB buffer.
- subBufferId – **[in]** Specify the selected sub-buffer.

static inline void XSPI_DisableAhbBufferPerfMonitor(XSPI_Type *base, uint8_t ahbBufferId)

Disable AHB buffer performance monitor for selected AHB buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the selected AHB buffer.

static inline void XSPI_StopAhbBufferPerfMonitor(XSPI_Type *base)

Stop AHB buffer performance monitor.

Parameters

- base – **[in]** XSPI peripheral base address.

static inline void XSPI_GetAhbBufferPerfMonitorResult(XSPI_Type *base, uint8_t ahbBufferId, *xspi_ahbBuffer_perf_monitor_result_t* *ptrPerfMonitorResult)

Get AHB buffer performance monitor result.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify AHB buffer Id.
- ptrPerfMonitorResult – **[out]** Pointer to the variable to store selected AHB buffer's performance monitor result.

static inline bool XSPI_CheckAhbBufferPerfMonitorTimeCounterOverflow(XSPI_Type *base)

Check if AHB buffer performance monitor timeout counter is overflow.

Parameters

- base – XSPI peripheral base address.

Return values

- false – AHB buffer performance monitor timeout counter is not overflow.
- true – AHB buffer performance monitor timeout counter is overflow.

static inline bool XSPI_CheckAhbBufferPerfMonitorHitOverflow(XSPI_Type *base, uint8_t ahbBufferId)

Check if AHB buffer performance monitor buffer hit counter is overflow.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the AHB buffer Id.

Return values

- false – Overflow not detected.
- true – Overflow is detected.

static inline bool XSPI_CheckAhbBufferPerfMonitorMissOverflow(XSPI_Type *base, uint8_t ahbBufferId)

Check if AHB buffer performance monitor buffer miss counter is overflow.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the AHB buffer Id.

Return values

- false – Overflow not detected.
- true – Overflow is detected.

static inline uint32_t XSPI_GetAhbBufferPerfMonitorTimeCounter(XSPI_Type *base)

Get AHB buffer performance monitor timeout value.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The value of time counter, in AHB clock cycles, since the performance monitor was running.

status_t XSPI_SetAhbAccessSplitSize(XSPI_Type *base, *xspi_ahb_split_size_t* ahbSplitSize)

Set AHB transaction(read/write) split size.

Note: For AHB write, if enable split feature(ahbSplitSize not set as kXSPI_AhbSplitSizeDisabled), XSPI will split one single AHB write burst into smaller bursts on external device side.

Note: For AHB read, if enable split feature, HW will realign the prefetch size to split size and in that way split a single read burst into smaller bursts.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbSplitSize – **[in]** Specify the AHB split size.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB access split size.

status_t XSPI_SetAhbAccessBoundary(XSPI_Type *base, *xspi_ahb_alignment_t* alignment)

Set AHB transaction(read/write) boundary.

Note: For AHB write, XSPI keeps track of the start address and then compares it with the address of subsequent transaction receive on the AHB bus, when the input alignment is reached, XSPI negates AHB HREADY to block new accesses. This signal will be kept low until all the previous received write data is written to external memory and chip select is

de-asserted. After that it allows next AHB write data to be received by making HREADY high thus ensuring that transaction is split at address boundary on external memory.

Note: For AHB read, XSPI checks the start address and end address to see if it is crossing the address boundary specified by input alignment. If the transaction crosses the address boundary, then it reduces the transaction size such that the data pre-fetch is stopped before the address boundary. Now if the AHB master is reading the data sequentially it will get buffer hits up to the input alignment boundary. When it reaches the next address boundary it will get a buffer miss and a new data pre-fetch will be launched towards the external memory device.

Parameters

- base – **[in]** XSPI peripheral base address.
- alignment – **[in]** Specify the AHB alignment, in type of xspi_ahb_alignment_t.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB access boundary.

status_t XSPI_SetAhbReadDataSeqId(XSPI_Type *base, uint8_t seqId)

Set AHB read sequence Id.

Parameters

- base – **[in]** XSPI peripheral base address.
- seqId – **[in]** Specify the sequence Id in LUT used for AHB read.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB read sequence Id.

status_t XSPI_SetAhbWriteDataSeqId(XSPI_Type *base, uint8_t seqId)

Set AHB write sequence Id.

Parameters

- base – **[in]** XSPI peripheral base address.
- seqId – **[in]** Specify the sequence Id in LUT used for AHB write.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted

- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB write sequence Id.

status_t XSPI_UpdateAhbHreadyTimeOutValue(XSPI_Type *base, uint16_t timeoutValue)

Specify how long XSPI can hold HEADY low while waiting for response to an AHB transfer.

Note: If the specified time out value expired, an AHB illegal transaction error will be triggerred.

Parameters

- base – **[in]** XSPI peripheral base address.
- timeoutValue – **[in]** In multiples of 50000 AHB clock cycles, default value is 3, it means waiting for 150000 AHB clock cycles.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_Success – Success to set AHB HREADY timeout value.

status_t XSPI_SetAhbErrorPayload(XSPI_Type *base, uint32_t highPayload, uint32_t lowPayload)

Pre-define error payload which will be provided on the read data bus when the HEADY timeout counter expires.

Note: If the incoming AHB read request master is not mapped to any of the AHB buffer, the XSPI will keep the AHB HREADY signal low and will provided this pre-defined error payload on the AHB read data bus when the HEADY timeout counter expires.

Parameters

- base – **[in]** XSPI peripheral base address.
- highPayload – **[in]** High 32bits payload to set.
- lowPayload – **[in]** Low 32bits payload to set.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_Success – Success to set AHB HREADY timeout value.

xspi_ahb_read_error_info_t XSPI_ReturnAhbReadErrorInfo(XSPI_Type *base)

Return AHB read error information.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns
Latest AHB read error information, in type of xspi_ahb_read_error_info_t.

static inline void XSPI_ClearAhbAccessSeqPointer(XSPI_Type *base)

Clear AHB access sequence pointer.

Parameters

- base – **[in]** XSPI peripheral base address.

void XSPI_GetAhbRequestSuspendInfo(XSPI_Type *base, *xspi_ahb_request_suspend_info_t* *ptrSuspendInfo)

Get Ahb request suspend information if priority mechanism is enabled.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrSuspendInfo – **[out]** Contain latest suspend info, the value may be 0xFF if the item is invalid in current suspend state.

static inline void XSPI_EnableAhbReadPrefetch(XSPI_Type *base, bool enable)

Enable/disable AHB Read Prefetch feature.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Used to enable/disable AHB read prefetch:
  - **false** Disable AHB read prefetch;
  - **true** Enable AHB read prefetch.

status_t XSPI_BlockAccessAfterAhbWrite(XSPI_Type *base, bool blockSequentWrite, bool blockRead)

Block access(write and read) after the AHB write operation.

Note: If external flash support RWW, read should not be blocked:

```
XSPI_BlockAccessAfterAhbWrite(XSPI0, true, false);
```

Note: If either of access is blocked, the page program wait flag will be asserted after an AHB write sequence completed. The assertion of flag will lock the arbitration and all access to the external memory are blocked. And the internal "page wait time" counter starts(Invoke XSPI_UpdatePageWaitTimeCounter to update counter value). After this counter reaches the value, a read is triggered by the XSPI module to read external device's status register(The seq id should be pre-defiend by XSPI_SetAhbReadStatusRegSeqId), and the value is stored in the XSPI internal regsiter. And there are two options(Invoke XSPI_SelectPPWFlagClearPolicy to select) to clear the asserted page program wait flag.

a. Automatic cleared by XSPI hardware;

b. Cleared by software.

Note:

As soon as the page program wait flag is asserted, another counter(SFP arbitration

lock counter) also started, if the flag not cleared in desired counter value the arbitration lock timeout error will be generated.

Parameters

- base – **[in]** XSPI peripheral base address.
- blockSequentWrite – **[in]** Block sequence write or not.
- blockRead – **[in]** Block read or not.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_Success – Success to set related registers..

status_t XSPI_SelectPPWFlagClearPolicy(XSPI_Type *base, *xspi_ppw_flag_clear_policy_t* policy)

Set Page program wait flag clear policy.

Note: If set as hardware policy, the device's WIP information should be set by invoking XSPI_SetSFMStatusRegInfo()

Parameters

- base – **[in]** XSPI peripheral base address.
- policy – **[in]** Specify the policy to clear page program wait flag.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set PPW flag clear policy.

status_t XSPI_UpdatePageWaitTimeCounter(XSPI_Type *base, uint32_t countValue)

Update page wait timeout counter.

Parameters

- base – **[in]** XSPI peripheral base address.
- countValue – **[in]** The value of counter, in AHB clock cycles.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_PageProgramWaitFlagAsserted – Page program wait flag already asserted.
- kStatus_Success – Successfully to update page wait timeout counter.

status_t XSPI_SetAhbReadStatusRegSeqId(XSPI_Type *base, uint8_t seqId)

Set AHB read status register sequence Id.

Note: The read status sequence only triggerred when the page wait time counter expired.

Parameters

- base – **[in]** XSPI peripheral base address.
- seqId – **[in]** Specify the sequence Id in LUT used for AHB read status register.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB read status register sequence Id.

static inline uint16_t XSPI_GetSFMStatusRegValue(XSPI_Type *base)

Get external serial flash's status register value.

Note: This function is only useful after the Page program wait flag is asserted due to either wait/read access is blocked after the AHB write operation.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The status regsiter value of external device.

status_t XSPI_SetSFMStatusRegInfo(XSPI_Type *base, *xspi_device_status_reg_info_t* *ptrStatusRegInfo)

Set Serial flash memory status register information.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrStatusRegInfo – **[in]** Pointer to the variable which contain status register information.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_PageProgramWaitFlagAsserted – Page program wait flag already asserted.
- kStatus_Success – Successfully to set status register information.

status_t XSPI_SetAhbAccessConfig(XSPI_Type *base, *xspi_ahb_access_config_t* *ptrAhbAccessConfig)

Set AHB access configuration.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrAhbAccessConfig – **[in]** Pointer to the variable which contains AHB access configurations.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB read status register sequence Id.

void XSPI_Cache64_SetRegionConfig(CACHE64_POLSEL_Type *base, *xspi_cache64_region_config_t* *config)

Set policy of cache regions.

Parameters

- base – CACHE64_POLSEL peripheral base address.
- config – Pointer to the structure xspi_cache64_region_config_t.

struct __xspi_device_status_reg_info

#include <fsl_xspi.h> The structure of external device's status register information(location, priority).

Note: Only useful when AHB write to flash device is enabled, and use hardware to clear ppw flag.

Public Members

uint8_t wipLocation

WIP bit field's location.

bool upperHalfWordSelected

Specify which half word(upper or lower) is selected.

bool value1Expired

Specify the value to be check o selected wip location to determine if the page progame wait period is expired, true means expected value is 1, false means expected value is 0.

union __xspi_dll_para

#include <fsl_xspi.h> The union of user custommed DLL parameters, in case of kXSPI_BypassMode bypassModePara is used, in case of kXSPI_AutoUpdateMode autoUpdateModoPara is used.

Public Members

struct *_xspi_dll_para* bypassModePara

struct *_xspi_dll_para* autoUpdateModoPara

struct _xspi_dll_config

#include <fsl_xspi.h> The structure of Delay-lock-loop configuration.

Public Members

xspi_dll_mode_t dllMode

Dll mode auto update mode or bypass mode

bool useRefValue

Specify to use reference value or not, if the reference code do not satisfy the specific usecase, it is application's responsibility to set custom parameters

- **true:** following inputs are ignored, use the reference values provided by NXP as instead;
- **false:** following inputs take effect.

xspi_dll_para_t dllCustomPara

User custommed parameter to configure DLL, different DLL mode corresponding to different parameters, please refer to the union xspi_dll_para_t.

uint8_t dllCustomDelayTapNum

User custommed number of delay elements for each delay tap.

bool enableCdl8

Enable course delay line 8.

struct _xspi_data_learning_config

#include <fsl_xspi.h> The configuration of data learning.

Note: Data learning feature only supported in the DQS sampling method(kXSPI_SampleClkFromExternalDQS).

Public Members

uint32_t pattern

Pre-defiend pattern to match.

bool deviceSupported

Specify if external device support data learning feature.

xspi_data_learning_pad_select_t padSelected

Used to select pad which use for pattern matching IO.

struct _xspi_sample_clk_config

#include <fsl_xspi.h> Sample clock configuration, which used to sample data from external device.

Public Members

xspi_sample_clk_source_t sampleClkSource
Specify the sample clock source.

bool enableDQSLatency
Enable DQS latency or not.

xspi_dll_config_t dllConfig
Specify the DLL configuration, to improve data accuracy, please adjust DLL settings based on specific use.

union __xspi__device__interface__settings
#include <fsl_xspi.h> The union of external device's settings, if external device following hyper bus protocol hyperBusSettings take effects, if external device following SPI protocol strandardExtendedSPISettings take effects.

Public Members

struct *_xspi_device_interface_settings* strandardExtendedSPISettings

struct *_xspi_device_interface_settings* hyperBusSettings

struct __xspi__device__ddr__config
#include <fsl_xspi.h> The structure of device DDR configuration.

Public Members

bool enableDdr
Used to enable/disable DDR mode.

xspi_ddr_data_aligned_clk_t ddrDataAlignedClk
Specify the DDR data aligned clock.

bool enableByteSwapInOctalMode
Swap byte order in octal mode.

struct __xspi__device__config
#include <fsl_xspi.h> External device configuration items.

Public Members

uint32_t xspiRootClk
XSPI serial root clock.

bool enableCknPad
Enable/disable clock on differential CKN pad for flash memory A.

xspi_device_interface_type_t deviceInterface
Type of external device's interface: hyperBus or Strandard/Extended SPI.

xspi_device_interface_settings_t interfaceSettings
Settings of specified interface.

uint8_t CSHoldTime
CS line hold time.

uint8_t CSSetupTime
CS line setup time.

xspi_sample_clk_config_t sampleClkConfig
Configuration of sample clock.

xspi_device_ddr_config_t *ptrDeviceDdrConfig
Set as NULL to set device as SDR mode, to change to DDR mode, this member should be populated.

xspi_device_addr_mode_t addrMode
Address mode of external device.

uint8_t columnAddrWidth
Width of column address.

bool enableCASInterleaving
Usually enabled in dual-die device.

uint32_t deviceSize[XSPI_SFAD_COUNT2]
Size of external device, the unit is KB.

xspi_device_status_reg_info_t *ptrDeviceRegInfo
Pointer to the structure to store external device's WIP register information, should set as NULL if AHB page write is not used or if AHB page write is used but use software policy to clear flag.

struct _xspi_mdad_config
#include <fsl_xspi.h> MDAD configuration.

Public Members

bool assignIsValid
Specify if the MDAD configuration for the target group is valid

bool enableDescriptorLock
This field provides a means to make the MDAD descriptor read-only. Once written '1' this field will remain '1' until hard reset

uint8_t maskType
0b - ANDed mask 1b - ORed mask

uint8_t mask
Defines the 6-bit mask value for the ID-Match comparison

uint8_t masterIdReference
Specifies the reference value of the Master-ID for MID-comparison

xspi_secure_attribute_t secureAttribute
Defines the secure attribute selection criteria for entry into descriptor queue

struct _xspi_sfp_mdad_config
#include <fsl_xspi.h> The structure of SFP MDAD configurations for all target groups.

Public Members

xspi_mdad_config_t tgMdad[(2U)]
Specify MDAD configurations for each target group.

struct _xspi_frad_config_t
#include <fsl_xspi.h> FRAD configuration.

Public Members

uint32_t startAddress

Specifies the specific flash memory region starting address

uint32_t endAddress

Specifies the specific flash memory region end address

uint8_t tg0MasterAccess

This field define the access restrictions for respective Master Domain corresponding to this FRAD region. Access permissions are decided based on secure an privilege attributes of current transaction. Read access is not restricted.

bool assignIsValid

This field indicates whether the FRAD Descriptor for a specific flash region is valid.

xspi_descriptor_lock_t descriptorLock

This field enables masking of accidental write on FRAD registers. Lock is enabled/disabled by Secure/ Privileged master.

xspi_exclusive_access_lock_mode_t exclusiveAccessLock

This field provides exclusive write lock over a FRAD region based on MDnACP.

struct __xspi__sfp__frad__config

#include <fsl_xspi.h> The structure of SFP FRAD configurations.

Public Members

xspi_frad_config_t fradConfig[(8U)]

Specify FRAD configuration for each item.

struct __xspi__frad__transaction__info

#include <fsl_xspi.h> The structure of latest FRAD transaction information.

Public Members

uint32_t startAddr

Latest IP access start address.

uint8_t masterId

Latest IP access master Id.

bool isSecureAccess

Latest IP access is secure access or not.

bool isPrivilegeAccess

Latest IP access is privilege access or not.

bool isCompError

Indicates the error status of the flash region specific comparision check for the latest transaction.

bool isCompValid

Indicates availability of the result or status of a flash region-specific comparison check.

struct __xspi__ip__access__config

#include <fsl_xspi.h> Configurations of IP access(including IP read and write).

Public Members

xspi_sfp_mdad_config_t *ptrSfpMdadConfig

Pointer to the variable in type of xspi_sfp_mdad_config_t to set SFP MDAD feature, set as NULL to disable SFP MDAD feature.

xspi_sfp_frad_config_t *ptrSfpFradConfig

Pointer to the variable in type of xspi_sfp_frad_config_t to set SFP FRAD feature, set as NULL to disable SFP FRAD

uint32_t sfpArbitrationLockTimeoutValue

Specify the time in IPS clock cycles before an unreleased arbitration triggers a timeout error.

uint32_t ipAccessTimeoutValue

Specify the maximum time in IPS clock cycles for XSPI to wait fro an ongoing write or read command to complete before terminating the command.

struct __xspi__tg__add__write__status

#include <fsl_xspi.h> The status of latest target group address write operation.

Public Members

uint8_t managerId

Manager Id of TG address write operation.

bool secureWrite

TG address write operation is secure write.

bool privilegedWrite

TG address write operation is privileged.

struct __xspi__transfer

#include <fsl_xspi.h> Transfer structure used for XSPI functional interface.

Public Members

uint32_t deviceAddress

Operation device address.

xspi_command_type_t cmdType

Execution command type.

uint8_t seqIndex

Sequence ID for command.

uint32_t *data

Data buffer.

size_t dataSize

Data size in bytes.

xspi_target_group_t targetGroup

Target group. include targetGroup[0] targetGroup[1]

struct __xspi__handle

#include <fsl_xspi.h> Transfer handle structure for XSPI.

Public Members

uint32_t state
: Internal state for XSPI transfer

uint8_t *data
: Data buffer.

size_t dataSize
: Remaining Data size in bytes.

size_t transferTotalSize
: Total Data size in bytes.

xspi_transfer_callback_t completionCallback
: Callback for users while transfer finish or error occurred

void *userData
: XSPI callback function parameter.

size_t curTransferSize
: Transfer size of current IP access.

struct _xspi_ahbBuffer_sub_buffer_config
: *#include <fsl_xspi.h>* The structure of sub-buffer configurations.

Public Members

xspi_subbuffer_division_factor_t divFactor
: Specify the divide factor for current sub-buffer.

uint32_t startAddr
: Specify the start address of current sub-buffer.

uint32_t endAddr
: Specify the end address of current sub-buffer.

bool enableAhbMonitor
: Enable/disable AHB performance monitor for current sub-buffer.

struct _xspi_ahbBuffer_config
: *#include <fsl_xspi.h>* Structure of AHB buffer configurations.

Note: When an AHB read access comes it is assigned to a buffer based on its master ID. Then it further assigned to a sub-buffer based on which sub-buffer address range this transaction lies into.

Note: When sub-buffer division is hit, in case of a buffer miss the controller will fetch the data equal to the size of that sub-buffer.

Note: If sub-buffer division is enabled for a buffer the hit/miss check is done at sub-buffer level. And if prefetch is enable, the prefetch takes place for individual sub-buffers.

Public Members

uint8_t masterId

Specify the ID of the AHB master to be associated with buffer.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer0Config

Pointer to sub buffer0's configuration.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer1Config

Pointer to sub buffer1's configuration.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer2Config

Pointer to sub buffer2's configuration.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer3Config

Pointer to sub buffer3's configuration.

struct _xspi_ahbBuffer_perf_monitor_result

#include <fsl_xspi.h> The result of AHB buffer performance monitor, including buffer miss count and buffer hit count.

Public Members

uint16_t bufferMissCount

Count of AHB read buffer miss events.

uint16_t bufferHitCount

Count of AHB read buffer hit events.

struct _xspi_ahb_write_config

#include <fsl_xspi.h> The configuration of AHB write access.

Public Members

bool blockSequenceWrite

Enable/disable sequence write operation after page program start.

bool blockRead

Enable/disable read operation after page program start, can be false for RWW extern device.

xspi_ppw_flag_clear_policy_t policy

Specify the policy to clear ppw flag, hardware or software.

uint32_t pageWaitTimeoutValue

Specify, in AHB clock cycles, how long XSPI waits before triggering a read of the device status register, only useful in hardare policy.

uint8_t AWRSeqIndex

Specify the sequence index for AHB write.

uint8_t ARDSRSeqIndex

Specify the sequence index for AHB read status, only useful if either sequence write or read is blocked.

struct _xspi_ahb_access_config

#include <fsl_xspi.h> The structure of AHB access configurations, including AHB buffer settings, AHB Read Seq ID, AHB write settings.

Public Members

xspi_ahbBuffer_config_t buffer[4]
AHB buffer size.

uint8_t ARDSeqIndex
Specify the sequence index for AHB read data.

bool enableAHBPrefetch
Enable/disable AHB read prefetch feature, when enabled, XSPI will fetch more data than current AHB burst.

bool enableAHBBufferWriteFlush
Set true to enable flush of AHB buffer, due to either AHB write or IP access to avoid stable data in AHB buffer.

xspi_ahb_split_size_t ahbSplitSize
Specify ahb split size, set as kXSPI_AhbSplitSizeDisabled if do not want to use this feature.

xspi_ahb_alignment_t ahbAlignment
AHB access towards flash is broken if this AHB alignment boundary is crossed, only support XSPI0 XSPI1

xspi_ahb_write_config_t *ptrAhbWriteConfig
Set as NULL if AHB write feature is not used.

struct __xspi__ahb__read__error__info
#include <fsl_xspi.h> The structure of information when AHB read error occur.

Public Members

uint32_t errorAddr
AHB read error address.

uint8_t errMasterId
AHB read error master Id.

struct __xspi__ahb__request__suspend__info
#include <fsl_xspi.h> The structure of information when AHB reqest is suspended.

Public Members

xspi_ahb_request_suspend_state_t state
Current AHB request state.

uint8_t subBufferId
Suspended AHB request's sub buffer Id.

uint8_t ahbBufferId
Suspended AHB request's buffer Id.

uint16_t dataLeft
How many data left to transfer.

uint32_t address
Current address of suspended AHB request.

struct __xspi__config
#include <fsl_xspi.h> XSPI configuration structure.

Public Members

xspi_byte_order_t byteOrder

Byte ordering endianness

bool enableDoze

Enable/disable doze mode support.

xspi_ahb_access_config_t *ptrAhbAccessConfig

Pointer to AHB access configuration, can be NULL is AHB access is not used.

xspi_ip_access_config_t *ptrIpAccessConfig

Pointer to IP access configuration, can be NULL is IP access is not used.

struct _xspi_cache64_config

#include <fsl_xspi.h> The structure of cache64 configuration.

Region0 range from base address to region0TopAddr[28:10] + 0x3FF, region1 range from region0TopAddr[28:10] + 0x400 to region1TopAddr[28:10] + 0x3FF, region1TopAddr[28:10] + 0x400 to end address.

Public Members

uint32_t region0TopAddr

Top address of region0, bit0-bit9 ignored.

xspi_cache64_policy_t region0Policy

Policy of region0.

uint32_t region1TopAddr

Top address of region1, bit0-bit9 ignored.

xspi_cache64_policy_t region1Policy

Policy of region1.

xspi_cache64_policy_t region2Policy

Policy of region2.

struct bypassModePara

Public Members

uint16_t delayElementCoarseValue

Delay element coarse adjustment value, range from 0 to 15.

uint16_t delayElementFineValue

Delay element fine adjustment value, range from 0 to 7.

uint16_t offsetDelayElementCount

Specify the number of offset delay elements, range from 0 to 15.

uint16_t enableHighFreq

If set as true, high frequency used for delay chain operations, otherwise low frequency is used.

uint16_t bypassModeReserved

Reserved.

struct autoUpdateModoPara

Public Members

uint16_t referenceCounter
: Specify count of interval for DLL phase detection, range from 0 to 15.

uint16_t resolution
: Specify minimum resolution for the DLL phase detctor, range from 0 to 15.

uint16_t offsetDelayElementCount
: Specify the number of offset delay elements, range from 0 to 15.

uint16_t tDiv16OffsetDelayElementCount
: Specify the number of T/16 offset elements in the incoming DQS, range from 0 to 15.

uint16_t enableHighFreq
: If set as true, high frequency used for delay chain operations, otherwise low frequency is used.

struct strandardExtendedSPISettings

Public Members

uint32_t pageSize
: The size of page to program, the unit is byte.

struct hyperBusSettings

Public Members

xspi_hyper_bus_x16_mode_t x16Mode
: Specify hyper bus X16 mode.

bool enableVariableLatency
: If enabled, the count of latency is depends on hyper bus device.

bool forceBit10To1
: Force bit 10 to logic one or not.

uint32_t pageSize
: The size of page to program, the unit is byte.

union enaPri

Public Members

bool enablePriority
: High Priority Enable, it can be written ‘1’ only when OTFAD is disabled, the AHB prefetch feature must be enabled if priority mechanism is enabled.

bool enableAllMaster
: When set, buffer3 acts as an all-master buffer.buff[i] routed to buffer3

struct ahbErrorPayload

Public Members

uint32_t highPayload
High 32bit error payload.

uint32_t lowPayload
Low 32bit error payload.

2.89 XSPI eDMA Driver

enum _xspi_edma_ntransfer_size
eDMA transfer configuration

Values:

enumerator kXSPI_EDMAnSize1Bytes
Source/Destination data transfer size is 1 byte every time

enumerator kXSPI_EDMAnSize2Bytes
Source/Destination data transfer size is 2 bytes every time

enumerator kXSPI_EDMAnSize4Bytes
Source/Destination data transfer size is 4 bytes every time

enumerator kXSPI_EDMAnSize8Bytes
Source/Destination data transfer size is 8 bytes every time

enumerator kXSPI_EDMAnSize32Bytes
Source/Destination data transfer size is 32 bytes every time

typedef struct *_xspi_edma_handle* xspi_edma_handle_t

typedef void (*xspi_edma_callback_t)(XSPI_Type *base, *xspi_edma_handle_t* *handle, *status_t* status, void *userData)
XSPI eDMA transfer callback function for finish and error.

typedef enum *_xspi_edma_ntransfer_size* xspi_edma_transfer_nsize_t
eDMA transfer configuration

FSL_XSPI_EDMA_DRIVER_VERSION
XSPI EDMA driver version 2.0.2.

void XSPI_TransferCreateHandleEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, *xspi_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle, *edma_handle_t* *rxDmaHandle)
Initializes the XSPI handle for transfer which is used in transactional functions and set the callback.

Parameters

- base – XSPI peripheral base address
- handle – Pointer to xspi_edma_handle_t structure
- callback – XSPI callback, NULL means no callback.
- userData – User callback function data.
- txDmaHandle – User requested DMA handle for TX DMA transfer.
- rxDmaHandle – User requested DMA handle for RX DMA transfer.

void XSPI_TransferUpdateSizeEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, *xspi_edma_transfer_nsize_t* nsize)

Update XSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

See also:

xspi_edma_transfer_nsize_t .

Parameters

- base – XSPI peripheral base address
- handle – Pointer to xspi_edma_handle_t structure
- nsize – XSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_EDMAnSize1Bytes(one byte).

status_t XSPI_TransferEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, *xspi_transfer_t* *xfer)

Transfers XSPI data using an eDMA non-blocking method.

This function writes/receives data to/from the XSPI transmit/receive FIFO. This function is non-blocking.

Parameters

- base – XSPI peripheral base address.
- handle – Pointer to xspi_edma_handle_t structure
- xfer – XSPI transfer structure.

Return values

- kStatus_XSPI_Busy – XSPI is busy transfer.
- kStatus_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer.
- kStatus_Success – XSPI successfully start edma transfer.

void XSPI_TransferAbortEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle)

Aborts the transfer data using eDMA.

This function aborts the transfer data using eDMA.

Parameters

- base – XSPI peripheral base address.
- handle – Pointer to xspi_edma_handle_t structure

status_t XSPI_TransferGetTransferCountEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, size_t *count)

Gets the transferred counts of transfer.

Parameters

- base – XSPI peripheral base address.
- handle – Pointer to xspi_edma_handle_t structure.
- count – Bytes transfer.

Return values

- kStatus_Success – Succeed get the transfer count.

- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

edma_handle_t *txDmaHandle
: eDMA handler for XSPI Tx.

edma_handle_t *rxDmaHandle
: eDMA handler for XSPI Rx.

size_t transferSize
: Bytes need to transfer.

xspi_edma_transfer_nsize_t nsize
: eDMA SSIZE/DSIZE in each transfer.

uint32_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 XSPI eDMA transfer.

xspi_edma_callback_t completionCallback
: A callback function called after the eDMA transfer is finished.

void *userData
: User callback parameter

struct _xspi_edma_handle
: *#include <fsl_xspi_edma.h>* XSPI DMA transfer handle, users should not touch the content of the handle.

Chapter 3

Middleware

3.1 Boot

3.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource

Overview

This repository is a fork of MCUboot (https://github.com/mcu-tools/mcuboot) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation

Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit MCUboot - Documentation to review details on the contents in this sub-repo.

Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution

Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream MCUboot project. Once this MCUboot fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

NXP Fork

This fork of MCUboot contains specific modifications and enhancements for NXP MCUXpresso SDK integration.

See *changelog* for details.

3.1.2 MCUboot



This is MCUboot version 2.2.0

MCUboot is a secure bootloader for 32-bits microcontrollers. It defines a common infrastructure for the bootloader and the system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software upgrade.

MCUboot is not dependent on any specific operating system and hardware and relies on hardware porting layers from the operating system it works with. Currently, MCUboot works with the following operating systems and SoCs:

- Zephyr
- Apache Mynewt
- Apache NuttX
- RIOT
- Mbed OS
- Espressif
- Cypress/Infineon

RIOT is supported only as a boot target. We will accept any new port contributed by the community once it is good enough.

MCUboot How-tos

See the following pages for instructions on using MCUboot with different operating systems and SoCs:

- Zephyr
- Apache Mynewt
- Apache NuttX
- RIOT
- Mbed OS
- Espressif
- *Cypress/Infineon*

There are also instructions for the *Simulator*.

Roadmap

The issues being planned and worked on are tracked using GitHub issues. To give your input, visit MCUboot GitHub Issues.

Source files

You can find additional documentation on the bootloader in the source files. For more information, use the following links:

- boot/bootutil - The core of the bootloader itself.
- boot/boot_serial - Support for serial upgrade within the bootloader itself.
- boot/zephyr - Port of the bootloader to Zephyr.
- boot/mynewt - Bootloader application for Apache Mynewt.
- boot/nuttx - Bootloader application and port of MCUboot interfaces for Apache NuttX.
- boot/mbed - Port of the bootloader to Mbed OS.
- boot/espressif - Bootloader application and MCUboot port for Espressif SoCs.
- boot/cypress - Bootloader application and MCUboot port for Cypress/Infineon SoCs.
- imgtool - A tool to securely sign firmware images for booting by MCUboot.
- sim - A bootloader simulator for testing and regression.

Joining the project

Developers are welcome!

Use the following links to join or see more about the project:

- Our developer mailing list
- Our Discord channel Get your invite

3.2 eIQ

3.2.1 eIQ

eIQ TensorFlow Lite for Micro Library User Guide

- *Overview*
- *TensorFlow Lite for Microcontrollers*
- *Build Status*
  - *Official Builds*
  - *Community Supported TFLM Examples*
  - *Community Supported Kernels and Unit Tests*
- *Contributing*
- *Getting Help*
- *Additional Documentation*

- *RFCs*

Overview TensorFlow Lite is an open source software library for running machine learning models on mobile and embedded devices. For more information, see www.tensorflow.org/lite.

For memory constrained devices, the library contains TensorFlow Lite for Microcontrollers. For more information, see *www.tensorflow.org/lite/microcontrollers*.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated TensorFlow Lite for Microcontrollers based on version 25-04-08 (from the 8th of April 2025 with commit). This document describes the steps required to download and start using the library. Additionally, the document describes the steps required to create an application for running pre-trained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

TensorFlow Lite for Microcontrollers TensorFlow Lite for Microcontrollers is a port of TensorFlow Lite designed to run machine learning models on DSPs, microcontrollers and other devices with limited memory.

Additional Links:

- Tensorflow github repository
- TFLM at tensorflow.org

Build Status

- GitHub Status

Official Builds

Build Type	Status
CI (Linux)	Run-CI passing
Code Sync	Sync from Upstream TF passing

Community Supported TFLM Examples This table captures platforms that TFLM has been ported to. Please see *New Platform Support* for additional documentation.

Platform	Status
Arduino	CI no status Arduino examples tests no status
Coral Dev Board Micro	TFLM + EdgeTPU Examples for Coral Dev Board Micro
Espressif Systems Dev Boards	CI failing
Renesas Boards	TFLM Examples for Renesas Boards
Silicon Labs Dev Kits	TFLM Examples for Silicon Labs Dev Kits
Sparkfun Edge	CI no status
Texas Instruments Dev Boards	

Community Supported Kernels and Unit Tests This is a list of targets that have optimized kernel implementations and/or run the TFLM unit tests using software emulation or instruction set simulators.

Build Type	Status
Cortex-M	Cortex-M passing
Hexagon	Run-Hexagon passing
RISC-V	RISC-V passing
Xtensa	Run-Xtensa passing
Generate Integration Test	Generate Integration Tests passing

Contributing See our *contribution documentation*.

Getting Help A Github issue should be the primary method of getting in touch with the TensorFlow Lite Micro (TFLM) team.

The following resources may also be useful:

1. SIG Micro email group and monthly meetings.
2. SIG Micro gitter chat room.
3. For questions that are not specific to TFLM, please consult the broader TensorFlow project, e.g.:
   - Create a topic on the TensorFlow Discourse forum
   - Send an email to the TensorFlow Lite mailing list
   - Create a TensorFlow issue
   - Create a Model Optimization Toolkit issue

Additional Documentation

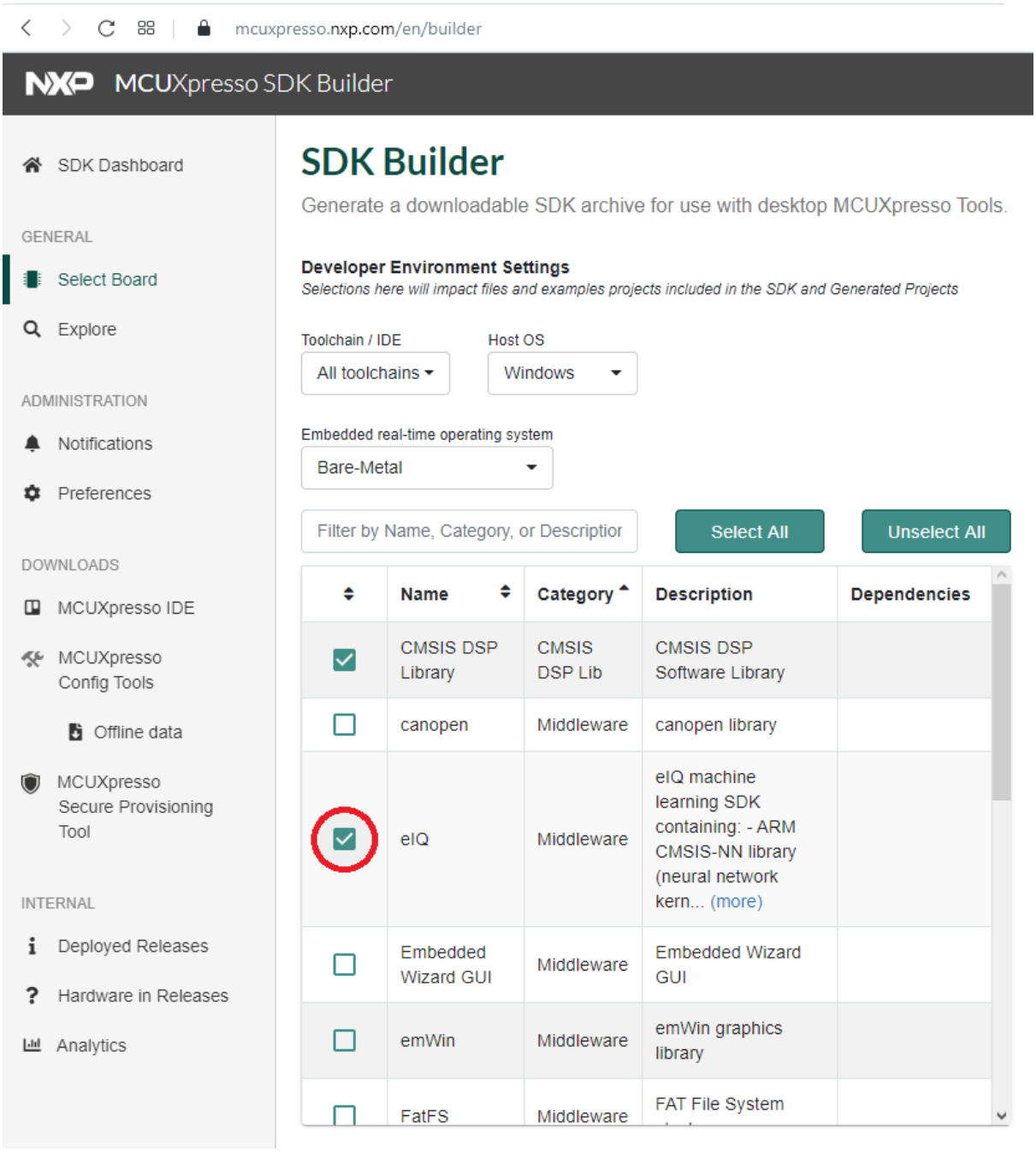
- *Continuous Integration*
- *Benchmarks*
- *Profiling*
- *Memory Management*
- *Logging*
- *Porting Reference Kernels from TfLite to TFLM*
- *Optimized Kernel Implementations*
- *New Platform Support*
- Platform/IP support
  - *Arm IP support*
- *Software Emulation with Renode*
- *Software Emulation with QEMU*

- *Python Dev Guide*
- *Automatically Generated Files*
- *Python Interpreter Guide*

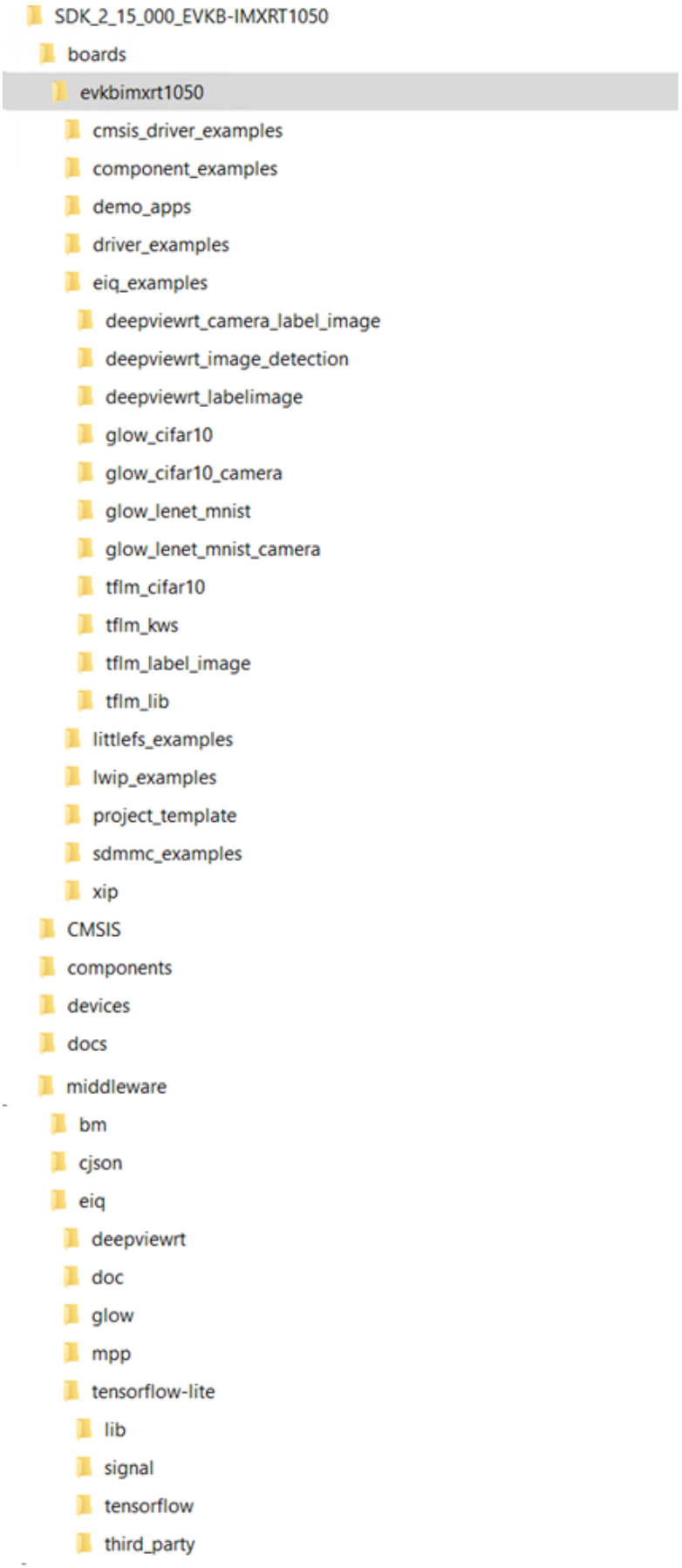
RFCs

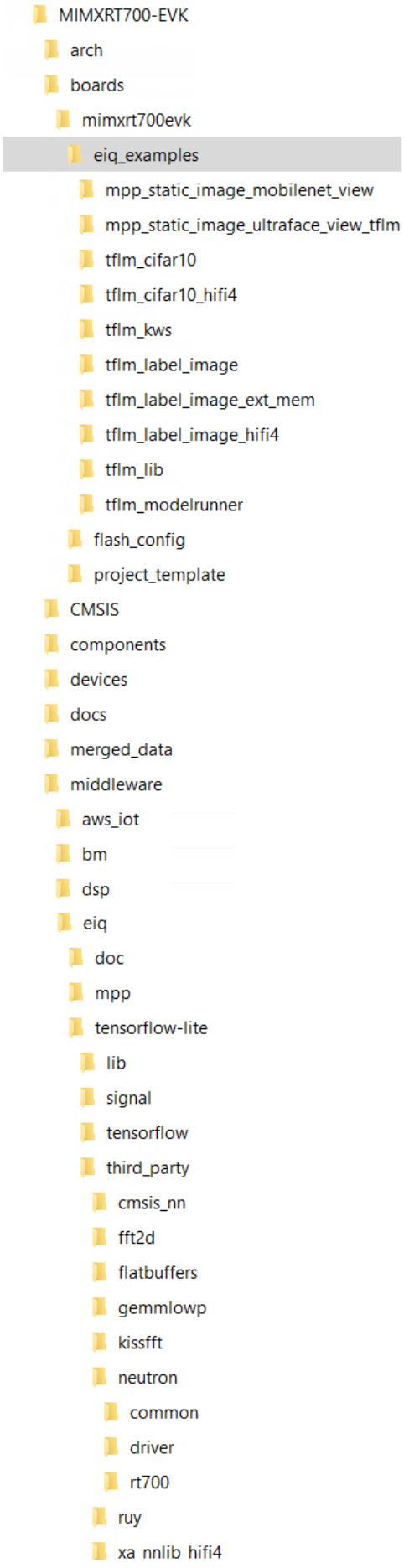
1. *Pre-allocated tensors*
2. *TensorFlow Lite for Microcontrollers Port of 16x8 Quantized Operators*

Deployment The eIQ TensorFlow Lite for Microcontrollers library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See *Figure 1*.



Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to *Figure 2*. The eIQ TensorFlow Lite library directories are highlighted in red.


SDK_2_15_000_EVKB-IMXRT1050
boards
evkbimxrt1050
cmsis_driver_examples
component_examples
demo_apps
driver_examples
eiq_examples
deepviewrt_camera_label_image
deepviewrt_image_detection
deepviewrt_labelimage
glow_cifar10
glow_cifar10_camera
glow_lenet_mnist
glow_lenet_mnist_camera
tflm_cifar10
tflm_kws
tflm_label_image
tflm_lib
littlefs_examples
lwip_examples
project_template
sdmmc_examples
xip
CMSIS
components
devices
docs
middleware
bm
cjson
eiq
deepviewrt
doc
glow
mpp
tensorflow-lite
lib
signal
tensorflow
third_party


MIMXRT700-EVK
arch
boards
mimxrt700evk
eiq_examples
mpp_static_image_mobilenet_view
mpp_static_image_ultraface_view_tflm
tflm_cifar10
tflm_cifar10_hifi4
tflm_kws
tflm_label_image
tflm_label_image_ext_mem
tflm_label_image_hifi4
tflm_lib
tflm_modelrunner
flash_config
project_template
CMSIS
components
devices
docs
merged_data
middleware
aws_iot
bm
dsp
eiq
doc
mpp
tensorflow-lite
lib
signal
tensorflow
third_party
cmsis_nn
fft2d
flatbuffers
gemmlowp
kissfft
neutron
common
driver
rt700
ruy
xa_nnlib_hifi4

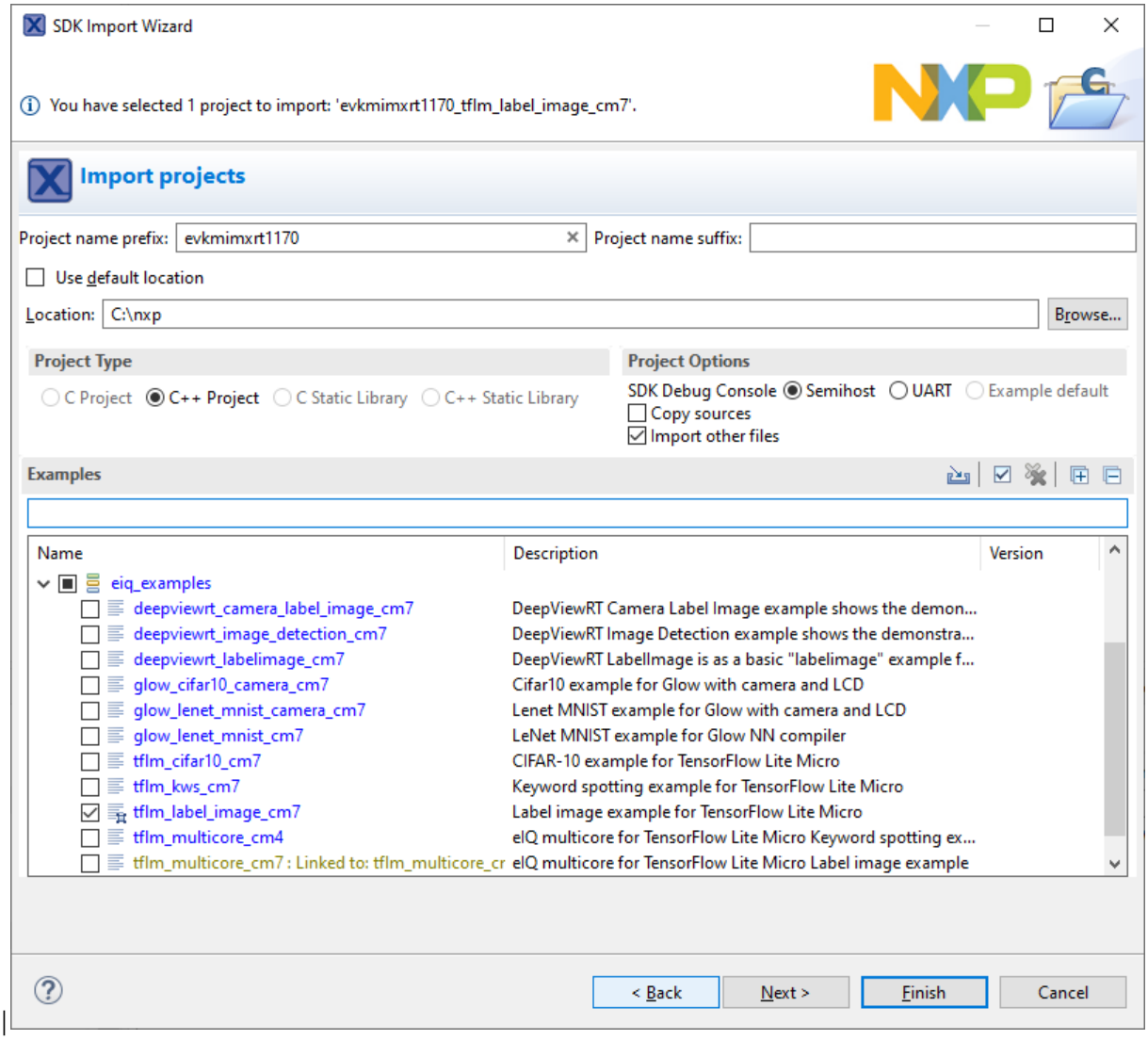
|

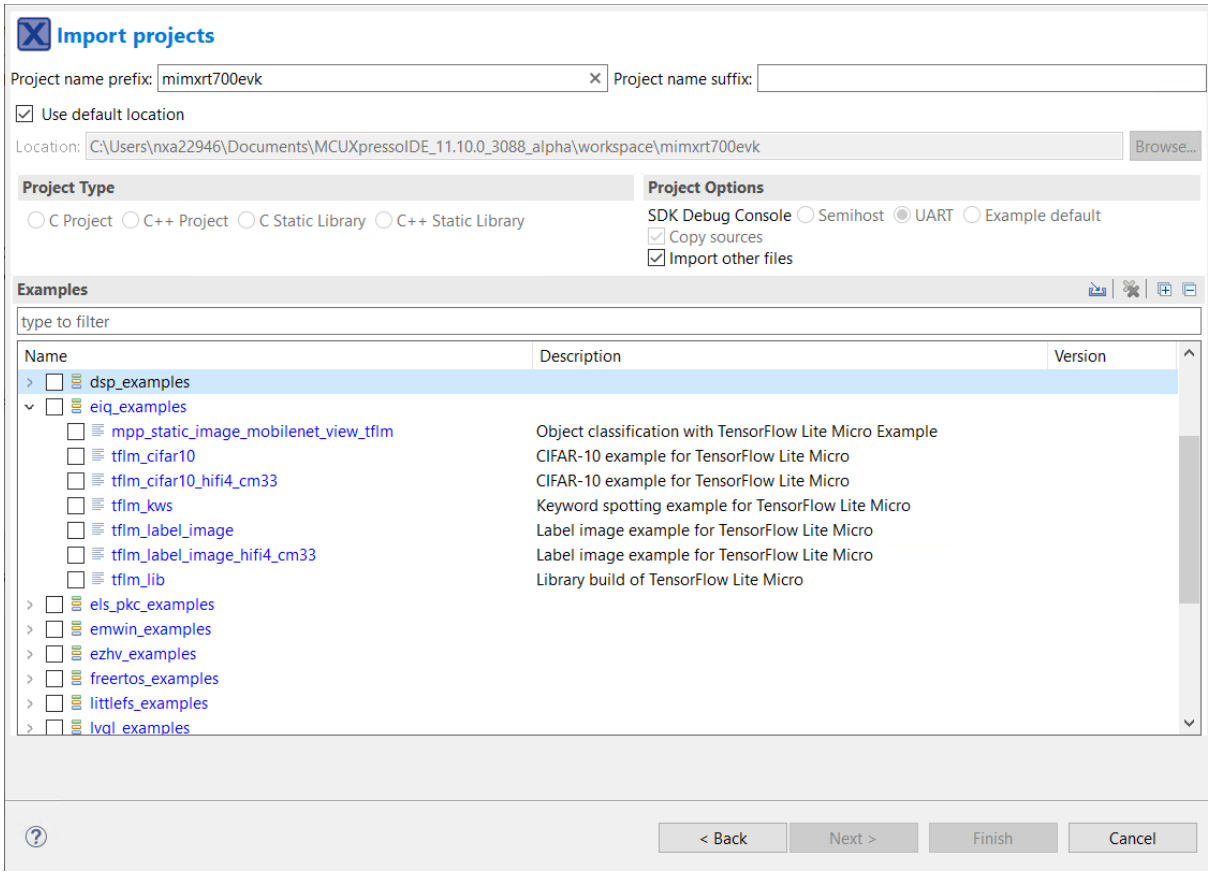
The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

Example applications The eIQ TensorFlow Lite library is provided with a set of example applications. For details, see *Table 1*. The applications demonstrate the usage of the library in several use cases.

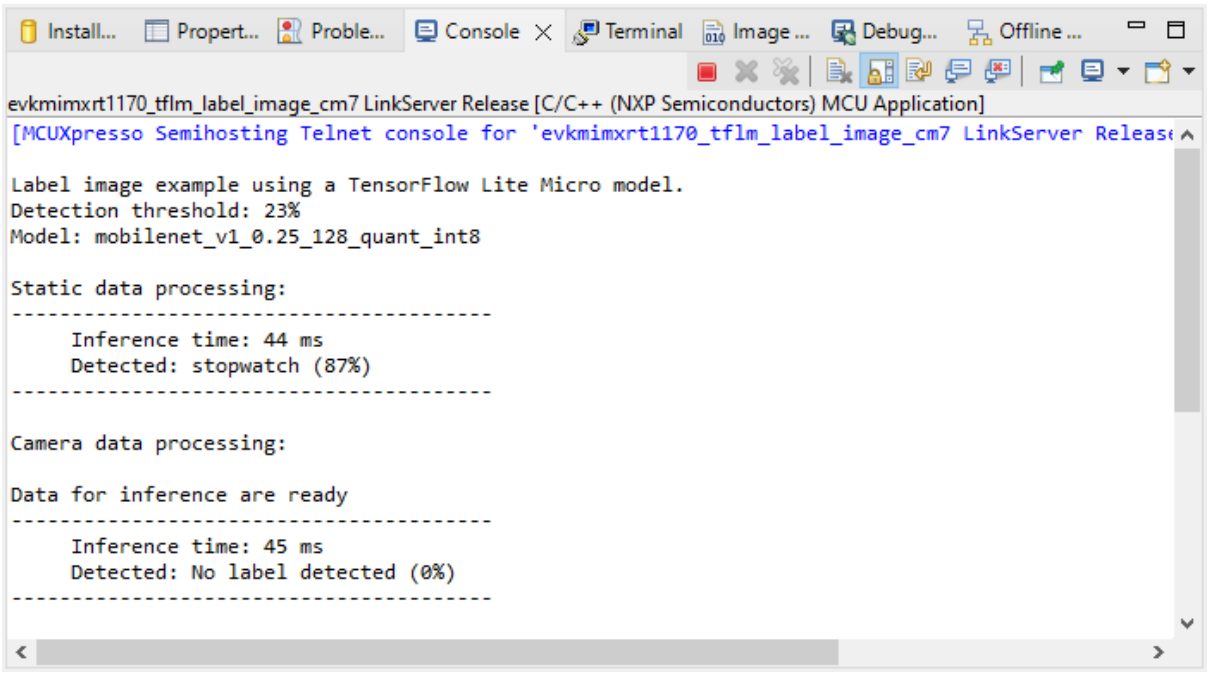
Name	Description	Availability
tflm_c	CIFAR-10 classification of 32 × 32 RGB pixel images into 10 categories using a small Convolutional Neural Network (CNN).	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_l	Keyword spotting application using a neural network for word detection in pre-processed audio input.	MCX-N947-EVK (no audio support) MCX-N947-FRDM (no audio support) MCX-N547-EVK (no audio support) MIMXRT700-EVK (no audio support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128 × 128 RGB pixel images into 1000 categorieswith eIQ Neutron NPU.	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 224 × 224 RGB pixel images into 1000 categorieswith eIQ Neutron NPU. In this example, it demonstrates how to fetch model's weight from external memory(xSPI flash) to internal SRAM for Neutron NPU execution.	MIMXRT700-EVK (no camera and display support)
tflm_c	CIFAR-10 classification of 32 × 32 RGBpixel images into 10 categories using a small Convolutional Neural Network. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for CIFAR-10 classification.	MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128 × 128 RGB pixel images into 1000 categories. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for image recognition application.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG). When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 1*.


SDK Import Wizard
You have selected 1 project to import: 'evkmimxrt1170_tflm_label_image_cm7'.
Import projects
Project name prefix: evkmimxrt1170
Project name suffix:
Use default location
Location: C:\nxp
Browse...
Project Type
C Project
C++ Project
C Static Library
C++ Static Library
Project Options
SDK Debug Console
Semihost
UART
Example default
Copy sources
Import other files
Examples
Name
Description
Version
eiq_examples
deepviewrt_camera_label_image_cm7
DeepViewRT Camera Label Image example shows the demon...
deepviewrt_image_detection_cm7
DeepViewRT Image Detection example shows the demonstra...
deepviewrt_labelimage_cm7
DeepViewRT LabelImage is as a basic "labelimage" example f...
glow_cifar10_camera_cm7
Cifar10 example for Glow with camera and LCD
glow_lenet_mnist_camera_cm7
Lenet MNIST example for Glow with camera and LCD
glow_lenet_mnist_cm7
LeNet MNIST example for Glow NN compiler
tflm_cifar10_cm7
CIFAR-10 example for TensorFlow Lite Micro
tflm_kws_cm7
Keyword spotting example for TensorFlow Lite Micro
tflm_label_image_cm7
Label image example for TensorFlow Lite Micro
tflm_multicore_cm4
eIQ multicore for TensorFlow Lite Micro Keyword spotting ex...
tflm_multicore_cm7 : Linked to: tflm_multicore_cr
eIQ multicore for TensorFlow Lite Micro Label image example
< Back
Next >
Finish
Cancel



After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, *Figure 2* shows the output of the tflm_label_image_cm7``tflm_label_image example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.



```
Label image example using a TensorFlow Lite Micro model.
Detection threshold: 23%
Model: mobilenet_v1_0.25_128_quant_int8_npu

Static data processing:
---------------------------------------
     Inference time: 3987 us
     Detected: stopwatch (87%)
---------------------------------------
```

|

Model Conversion to TensorFlow Lite Format The eIQ® Toolkit provides a comprehensive end-to-end environment for machine learning (ML) model development and deployment. Designed for NXP EdgeVerse processors, the toolkit includes both an intuitive GUI-based tool (eIQ Portal) and command-line utilities for advanced workflows.

One key component, the eIQ ModelTool, enables seamless conversion of ML models from popular formats such as TensorFlow, PyTorch, and ONNX into the TensorFlow Lite (TFLite) format. These converted models can be further optimized and deployed on NXP platforms for inference acceleration.

Model Conversion for NXP eIQ Neutron NPU To leverage the NXP eIQ Neutron NPU for hardware acceleration, models must undergo additional processing using the Neutron Converter Tool. This tool transforms standard quantized TensorFlow Lite models into a format optimized for execution on the Neutron NPU.

The key steps involved in this process are as follows:

1. Convert to Quantized TensorFlow Lite Model: Ensure the model is in a quantized TFLite format before running the Neutron Converter.
2. Run the Neutron Converter Tool: The Neutron Converter analyzes the TFLite model, identifies supported operators, and replaces them with specialized NPU-compatible nodes. Unsupported operations are executed using fallback mechanisms, such as:

- CMSIS-NN for optimized CPU execution
- Reference Operators for unsupported cases

3. Execute on Target Platform: The converted model runs efficiently on the Neutron NPU using a custom TFLite Micro-operator implementation.

Example: Converting a Quantized TensorFlow Lite Model for Neutron NPU The following is a sample command-line invocation for the Neutron Converter tool:

```
neutron-converter --input mobilenet_v1_0.25_128_quant.tflite \
                  --output mobilenet_v1_0.25_128_quant_npu.tflite \
                  --target imxrt700 \
                  --dump-header-file-output
```

Note: This will convert the source tflite model to neutron compatable model, meanwhile, it will dump the model as one headfile name as “mobilenet_v1_0.25_128_quant_npu.h”.

Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE This section lists the steps to Prepare CM33 Core for the examples and Prepare DSP core for the examples.

Prepare CM33 Core for the examples

1. The tflm_cifar10_hifi4 and tflm_label_image_hifi4 examples consist of two separate applications that run on the CM33 core0 and DSP core. The CM33 core0 application initializes the DSP core and starts it.

   To debug the application:

   1. Set up and execute the CM33 application using an environment of your choice.
   2. Build and execute the examples located in:

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/cm33/

      <SDK_ROOT>/ boards/mimxrt700evk/eiq_examples/tflm_label_image_hifi4/cm33/

2. The tflm_cifar10_hifi1 example consists of three separate applications that run on the CM33 core0, CM33 core1, and DSP core. The CM33 core0 application initializes the CM33 core1 core and starts it. The CM33 core1 application initializes the DSP core and starts it.

   To debug the application:

   1. Set up and build the CM33 core1 application using an environment of your choice.
   2. Set up and execute the CM33 core0 application using an environment of your choice.
   3. Build and execute the example located in:

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core1/

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core0/

      Note: ARMGCC toolchain and IAR Embedded Workbench are both supported. To enable compatibility with RT700, IAR Embedded Workbench may require a patch. There are default DSP core images in the SDK. For details on how to build the examples, refer to Prepare DSP core for the examples.

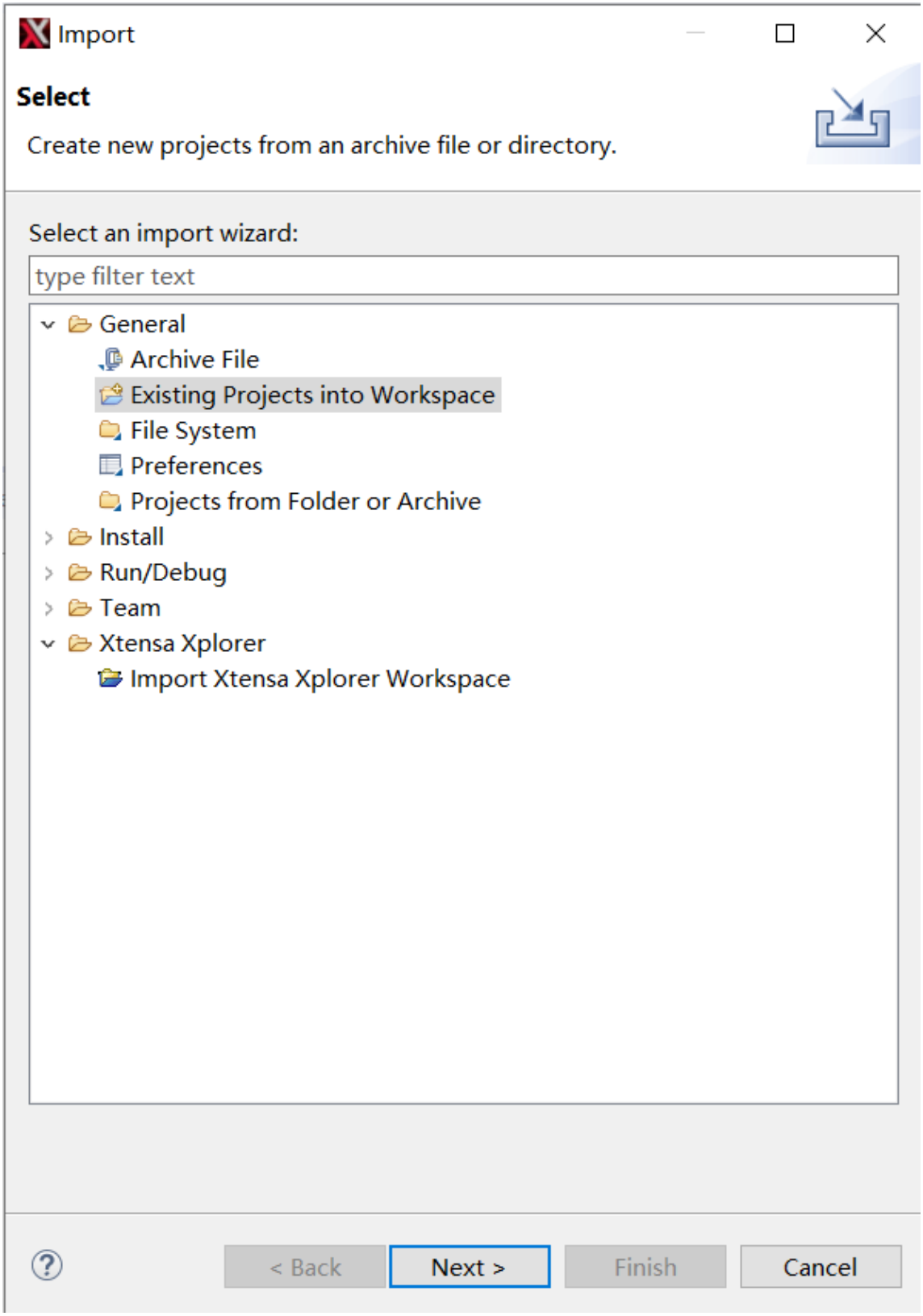
Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

Prepare DSP core for the examples

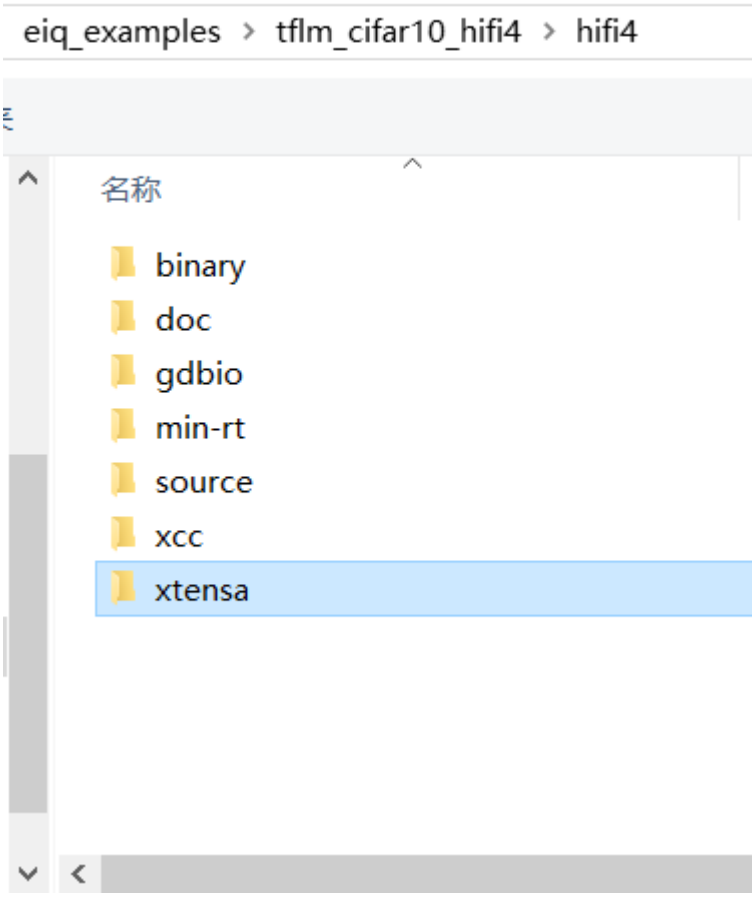
The projects for different supported toolchains are built. The “xcc” project builds on the command line and the “xtensa” directory is an Xplorer IDE project.

To run the tflm_cifar10_hifi4 example, import the SDK sources into the Xplorer IDE.

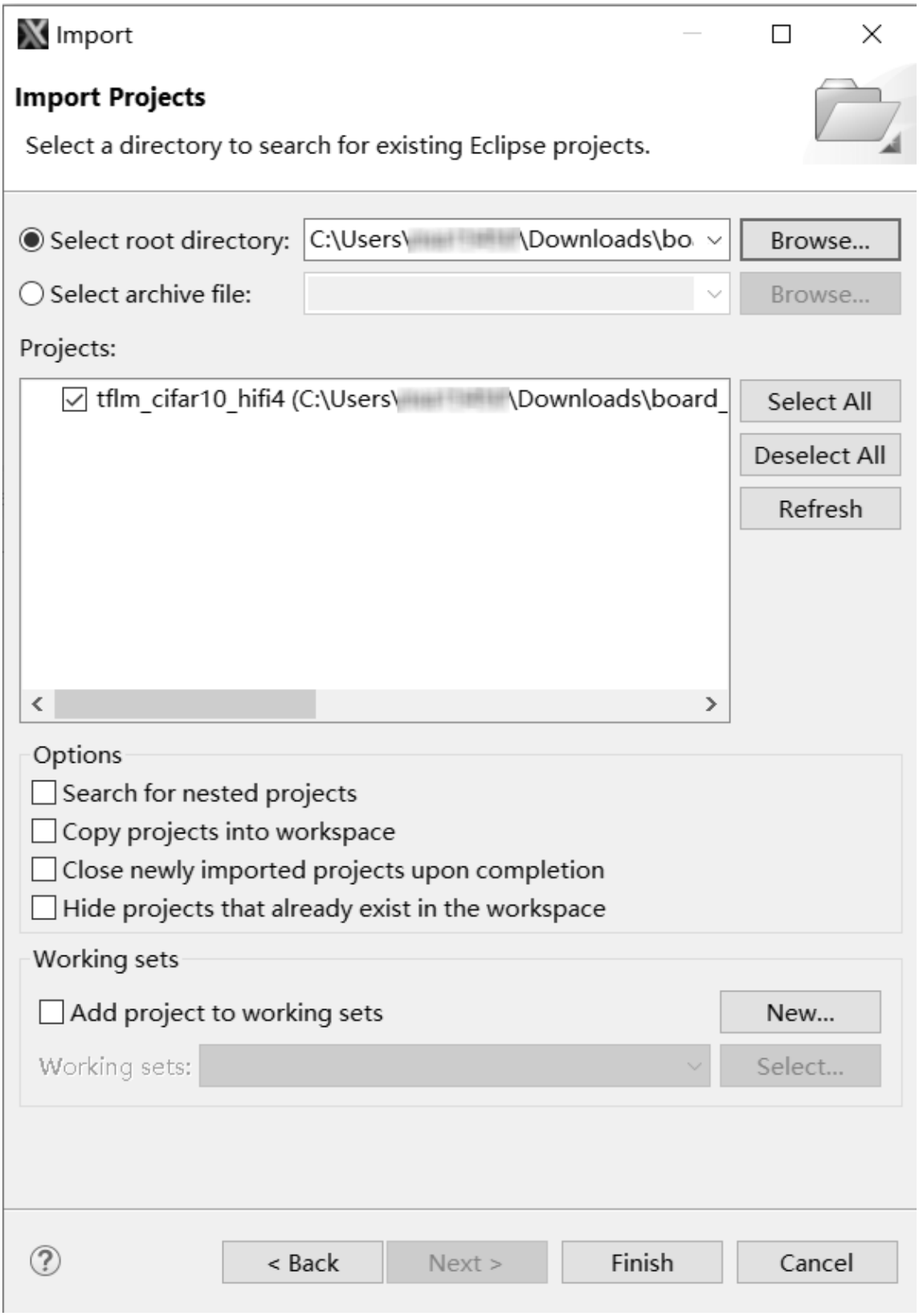
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Click **Next**.
3. Select the SDK directory/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/xtensa as the root directory.

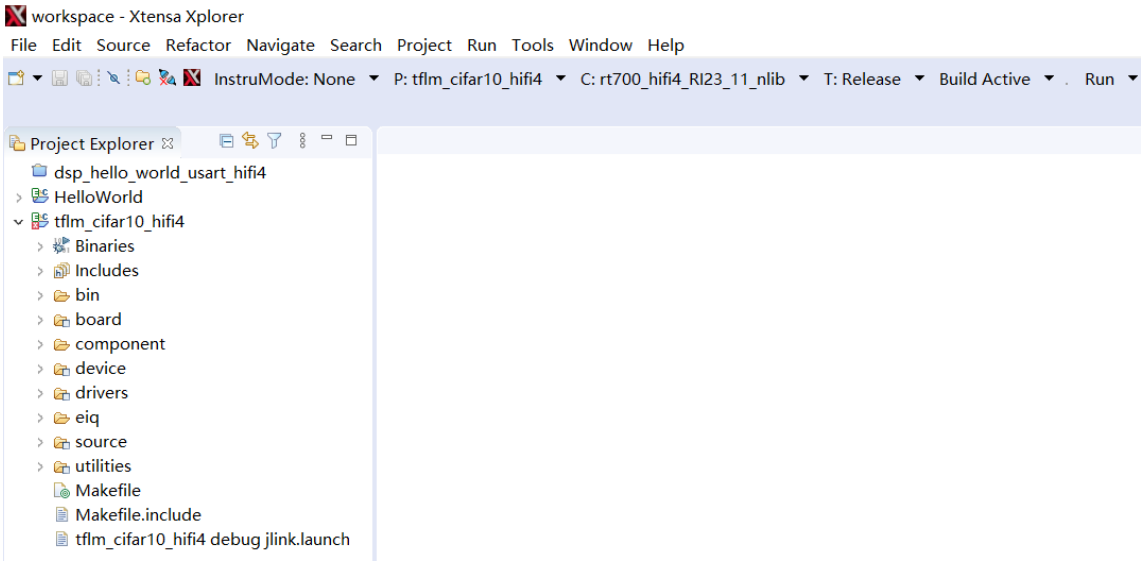


4. Click **Select Folder**.
5. Leave all the other options check boxes blank.

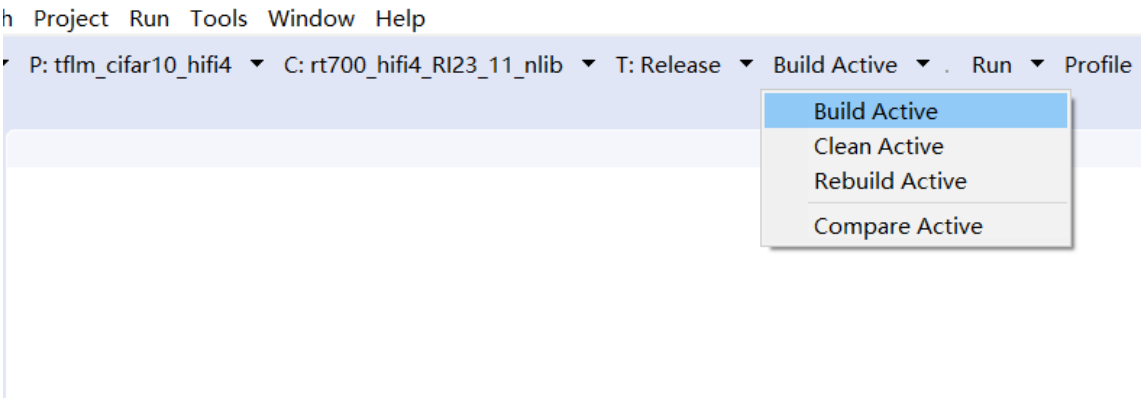


Once imported, the tflm_cifar10_hifi4 example appears in the **Project Explorer**.

6. To make a build selection for the project and hardware target configuration, use the drop-down buttons on the menu bar.



7. To build the DSP application image for the CM33 application, select the **Release target** option in the Xplorer IDE as below.



8. Three DSP binaries are generated and are loaded into different TCM or SRAM address segments:
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_data_release.bin
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_literal_release.bin
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_text_release.bin

Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

Running an inference After converting the model to the TensorFlow Lite format, it is converted into a C language array to include it in the application source code. The *xxd* utility can be used for this purpose (distributed with the *Vim* editor for many platforms on https://www.vim.org/) as shown in *Converting a model to a C language header file*. The utility converts a TensorFlow Lite model into a C header file with an array definition containing the binary image of the model and a variable containing the data size.

Converting a model to a C language header file {#EXAMPLE_4 .section}

```
xxd -i mobilenet_v1_0.25_128_quant.tflite > mobilenet_v1_0.25_128_quant_model.h
```

After the header file is generated, the type of the array is changed from unsigned char to const char to match the library API input parameters and the default array name can be changed to a more convenient one. The user must align the buffer to at least 64-bit boundary (the size of a double-precision floating-point number) to avoid misaligned memory access. The alignment can be achieved by using the __ALIGNED(16) macro from the cmsis_compiler.h header file (available in the MCUXpresso SDK) in the array declaration before the data assignment.

The easiest way to create an application with the proper configuration is to copy and modify an existing example application. To learn where to find the example applications and how to build them, see the *Example applications*.

Running an inference using TensorFlow Lite for Microcontrollers involves several steps (shown for quantized model with signed 8-bit values as input and 32-floating point values as output):

1. Include the necessary eIQ TensorFlow Lite Micro library header files and the converted model.

   Including header files

   ```
   #include "tensorflow/lite/micro/micro_error_reporter.h"
   #include "tensorflow/lite/micro/micro_interpreter.h"
   #include "tensorflow/lite/micro/all_ops_resolver.h"
   #include "mobilenet_v1_0.25_128_quant_model.h"
   ```

2. Allocate a static memory buffer for input and output tensors and intermediate arrays. Load the FlatBuffer model image (assuming the mobilenet_v1_0.25_128_quant_model.h file generated in *Converting a model to a C language header file* defines an array named mobilenet_model and a size variable named mobilenet_model_len), build the interpreter object and allocate memory for tensors.

   Loading the FlatBuffer model

   ```
   constexpr int kTensorArenaSize = 1024 * 1024;
   static uint8_t tensorArena[kTensorArenaSize];
   const tflite::Model* model = tflite::GetModel(mobilenet_model);
   // TODO: Report an error if model->version() != TFLITE_SCHEMA_VERSION
   static tflite::AllOpsResolver microOpResolver;
   static tflite::MicroErrorReporter microErrorReporter;
   static tflite::MicroInterpreter interpreter(model,
     microOpResolver, tensorArena, kTensorArenaSize,
     microErrorReporter);
   interpreter->AllocateTensors();
   // TODO: Check return value for kTfLiteOk
   ```

3. Fillhe input data into the input tensor. For example, if a speech recognition model, image data from a camera or audio data from a microphone. The dimensions of the input data must be the same as the dimensions of the input tensor. These dimensions were specified when the model was created.

   Fill-in input data

   ```
   // Get access to the input tensor data
   TfLiteTensor* inputTensor = interpreter->input(0);
   // Copy the input tensor data from an application buffer
   for (int i = 0; i < inputTensor->bytes; i++)
     inputTensor->data.int8[i] = input_data[i];
   ```

4. Run the inference and read the output data from the output tensor. The dimensions of the output data must be the same as the dimensions of the output tensor. These dimensions were specified when the model was created.

   Running inference and reading output data

```
// Run the inference
interpreter->Invoke();
// TODO: Check the return value for TfLiteOk
// Get access to the output tensor data
TfLiteTensor* outputTensor = interpreter->output(0);
// Copy the output tensor data to an application buffer
for (int i = 0; i < outputTensor->bytes / sizeof(float32); i++)
  output_data[i] = outputTensor->data.f[i];
```

NPU inference {#npu_infer .section} Running an inference using a model converted for the NPU requires registration of a custom operator implementation. First the header file with the custom operator implementation interface must be included.

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/kernels/neutron/neutron.h"
```

Next, the specialized implemetation has to be registered in the operator resolver object.

```
static tflite::AllOpsResolver microOpResolver;
microOpResolver.AddCustom(tflite::GetString_NEUTRON_GRAPH(),
      tflite::Register_NEUTRON_GRAPH());
```

The specialized NPU nodes from the converted model are the executed using this newly registered implementation.

Adjusting the tensor arena size {#adjust_arena .section} The tensor arena is a static memory buffer used for intermediate tensor and scratch buffer allocation. The size of the tensor arena buffer is set by the kTensorArenaSize constant in the example above. The value depends on the tensor sizes used in the model and on the hardware-specific implementations of kernels, which may require various sizes of scratch buffers for intermediate computations. The value can be determined experimentally by running an inference with a small value, so the library fails with an insufficient tensor memory error and prints the missing amount. Continue adjusting the size until the error stops being reported. If the target hardware changes, readjust the value.

Code size optimization Typically, models do not use all the operators that are available in TensorFlow Lite. However, because of the default operator registration mechanism used in the library, the toolchain linker is not able to remove the code of unused operators. In order to reduce code size, it is possible to only register the specific operators used by a model. To determine which operators are used by a particular model, a model visualizer tool like Netron can be used. Then a mutable operator resolver object can be created that only registers the operators that are used by the model being inferenced.

Use the tflite::MicroMutableOpResolver object template, which is later passed to the tflite::MicroInterpreter object. Depending on the list of used operators, the result should be similar to the following code snippet. Make sure to update the MicroMutableOpResolver template parameter to reflect the number of operators that need to be registered.

Register only used operators in TensorFlow Lite Micro {#SECTION_SS1_DJQ_QPB .section}

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
      #include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
      tflite::MicroMutableOpResolver<6> microOpResolver;
      microOpResolver.AddAveragePool2D();
      microOpResolver.AddConv2D();
      microOpResolver.AddDepthwiseConv2D();
```

(continues on next page)

(continued from previous page)

```
microOpResolver.AddDequantize();
microOpResolver.AddReshape();
microOpResolver.AddSoftmax();
static tflite::MicroInterpreter interpreter(
model, microOpResolver, tensorArena, kTensorArenaSize, microErrorReporter);
```

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

eIQ ExecuTorch Library User Guide

Overview ExecuTorch is an end-to-end solution for enabling on-device inference capabilities across mobile and edge devices including wearables, embedded devices and microcontrollers. It is part of the PyTorch Edge ecosystem and enables efficient deployment of PyTorch models to edge devices. For more information, see https://pytorch.org/executorch-overview.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated ExecuTorch based on version v0.5.0 with initial support for Neutron Backend. Neutron Backend enables acceleration of ML models on the eIQ® Neutron Neural Processing Unit (NPU).

This document describes the steps required to download and start using the ExecuTorch. Additionally, the document describes the steps required to create an application for running pretrained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

Supported platforms:

- i.MX RT700

Installation The ExecuTorch, with the Neutron Backend consists of:

- ExecuTorch with Neutron Backend for Ahead of Time ML Model Compilation
- Neutron Converter
- MCUXpresso SDK

Here we briefly describe each components purpose and steps to install them.

The **ExecuTorch AoT** and **Neutron Converter** are needed to convert a PyTorch model to ExecuTorch and Delegate it to eIQ Neutron NPU using the Neutron Backend. The **MCUXpresso SDK** provides project to build the ExecuTorch Runtime Library, the example application with simple CNN, toolchains and other middleware libraries to build and deploy the application on the target platform.

If you want run to prepared example application on the i.MX RT700 platform, and skip the model preparation phase continue with the *MCUXpresso SDK Part*.

ExecuTorch for Ahead of Time model preparation The ExecuTorch enables to deploy PyTorch models on edge devices. For this purpose the PyTorch model must be processed and converter by the ExecuTorch Ahead of Time (AoT) part. You can obtain the full ExecuTorch including the AoT part aligned with this version of MCUX SDK from the mcuxsdk-middleware-executorch release/mcux-full branch.

Installation Prerequisities:

- x86 Linux Machine with GLIBC-2.29 or higher (e.g. Ubuntu 20.04 or higher)
- Python 3.10, 3.11 or 3.12

To build and install the ExecuTorch follow these steps:

1. (Optional) Setup python virtual environment on desired location and activate it.

```
$ python3 -m venv venv
$ source venv/bin/activate
```

2. Clone the ExecuTorch from mcuxsdk-middleware-executorch

```
$ git clone --branch release/mcux-full https://github.com/nxp-mcuxpresso/mcuxsdk-middleware-executorch.git
$ cd mcuxsdk-middleware-executorch
$ git submodule update --init --recursive
```

3. Build and install the ExecuTorch and its dependencies:

```
$ ./install_requirements.sh
```

[!WARNING] The install_requirements.sh installs the CPU version of torch from https://download.pytorch.org/whl/cpu. If you are behind corporate proxy, it might have issues accessing it and you will see warnings like:

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,␣
↪status=None)) after connection broken by 'SSLError(SSLCertVerificationError(1, '[SSL:␣
↪CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer␣
↪certificate (_ssl.c:1006)'))': /whl/test/cpu/torch/
```

In this case the CUDA version of torch is installed and the install_requirements.sh script fails with:

```
PyTorch: CUDA cannot be found.  Depending on whether you are building
```

Make sure the pip can access the https://download.pytorch.org/whl/cpu PyPI.

Next continue with installation of the *Neutron Converter*

Neutron Converter The eIQ Neutron Backend uses the Neutron Converter to convert the ExecuTorch program to the eIQ Neutron NPU microcode.

Installation The Neutron Converter is available as a Python package and can be installed by the pip command from eiq.nxp.com/repository:

```
pip install --index-url https://eiq.nxp.com/repository neutron_converter_SDK_25_09==1.0.0
```

The Neutron Converter is used internally by the ExecuTorch, and it is tied to the particular BSP you are using - the suffix of the python package name. In the code snippet above the flavor is the SDK_25_09. In the *aot_neutron_convert.py* example script by the --neutron_converter_flavor parameter.

MCUXpresso SDK The MCUXpresso SDK is used to build, debug and deploy the application using the ExecuTorch on the target platform.

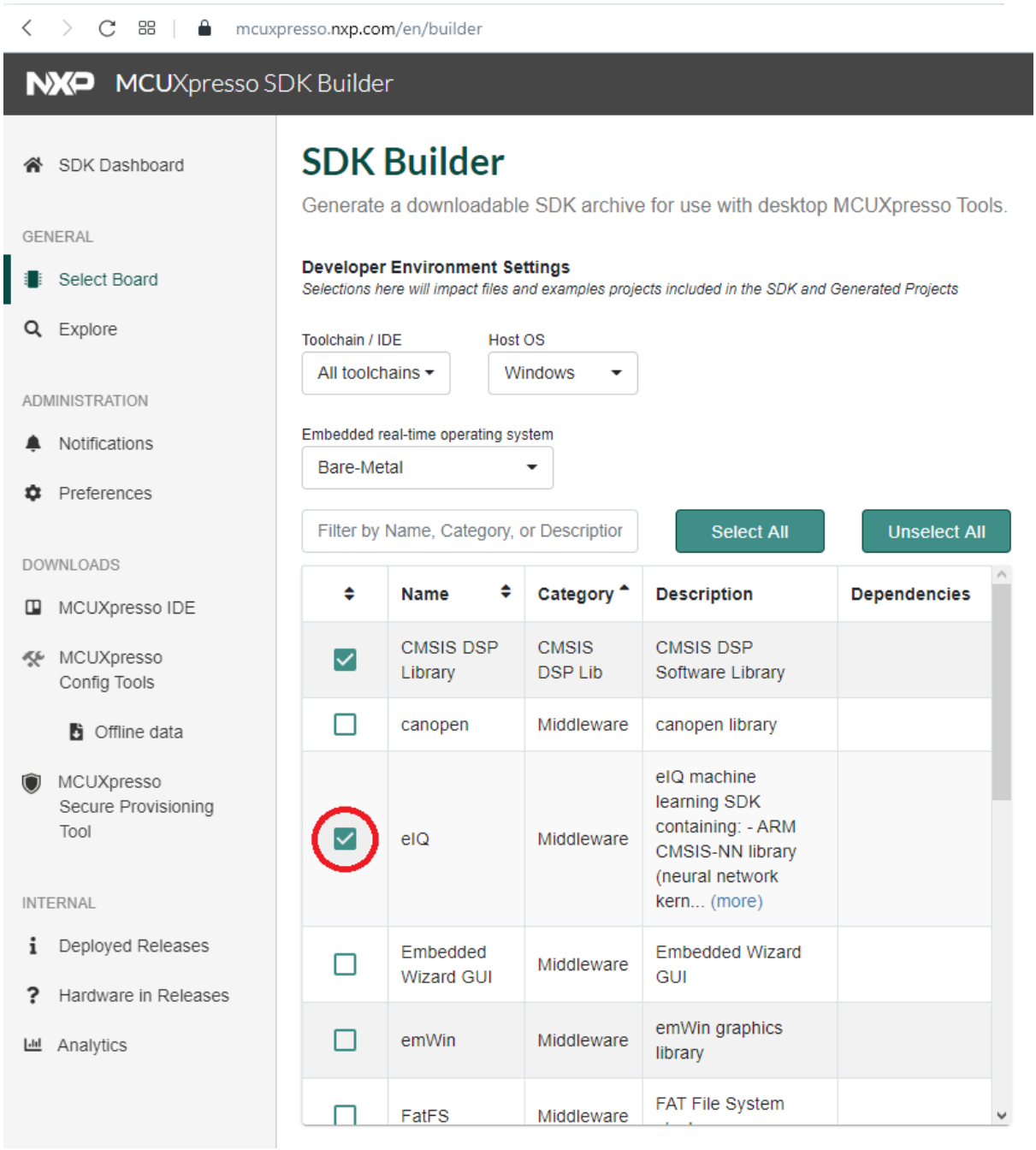
You can obtain the MCUXpresso SDK from MCUXpresso SDK Builder including the IDE. See the *getting_mcuxpress* for details.

In the MCUXpresso SDK, there are 2 projects available related to ExecuTorch:

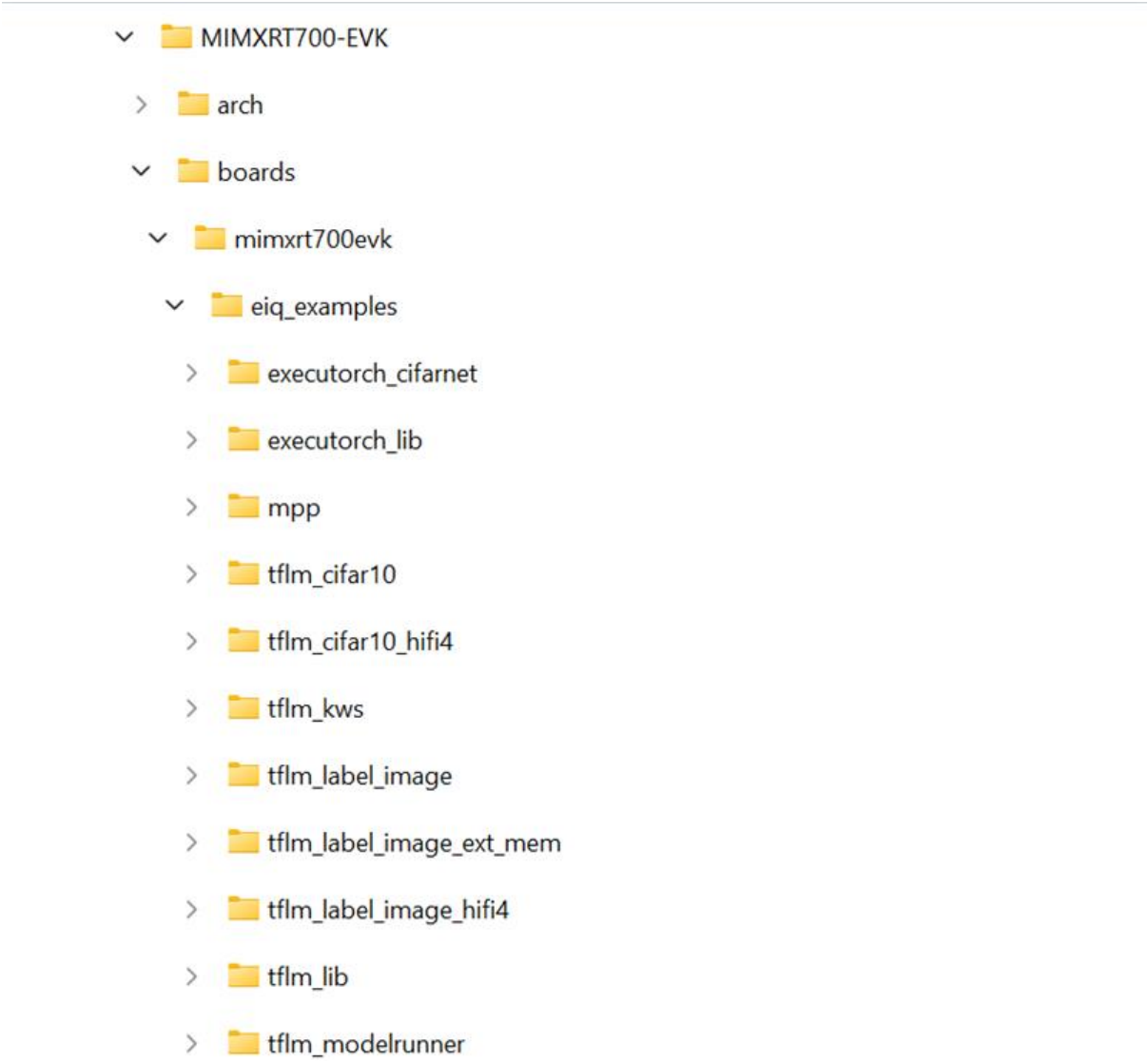
- executorch_lib
- executorch_cifarnet

For more details see *example_applications*. Here you will find the details to run build and run the demo applications.

Getting the MCUXpresso SDK with eIQ ExecuTorch The eIQ ExecuTorch library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See *Figure 1*.



Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to *Figure 2*.



The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

PyTorch Model Conversion to ExecuTorch Format In this guideline we will show how to use the ExecuTorch AoT part to convert a PyTorch model to ExecuTorch format and delegate the model computation to eIQ Neutron NPU using the eIQ Neutron Backend.

First we will start with an example script converting the model. This example show the CifarNet model preparation. It is the same model which is part of the example_cifarnet

The steps are expected to be executed from the executorch root folder, in our case the mcuxsdk-middleware-executorch

1. After building the ExecuTorch you shall have the libquantized_ops_aot_lib.so located in the pip-out folder. We will need this library when generating the quantized cifarnet ExecuTorch model. So as first step we will find it:

```
$ find ./pip-out -name 'libquantized_ops_aot_lib.so'
./pip-out/temp.linux-x86_64-cpython-310/cmake-out/kernels/quantized/libquantized_ops_aot_lib.so
./pip-out/lib.linux-x86_64-cpython-310/executorch/kernels/quantized/libquantized_ops_aot_lib.so
```

2. Now run the aot_neutron_compile.py example with the cifar10 model

```
$ python examples/nxp/aot_neutron_compile.py \
  --quantize --so_library ./pip-out/lib.linux-x86_64-cpython-310/executorch/kernels/quantized/libquantized_
↪ops_aot_lib.so \
  --delegate --neutron_converter_flavor SDK_25_09 -m cifar10
```

3. It will generate you cifar10_nxp_delegate.pte file which can be used with the MXUXpresso SDK cifarnet_example project.

The generated PTE file is used in the executorch_cifarnet example application, see *example_application*.

MCUXpresso SDK Example applications The MCUXpresso SDK provides a set of projects and example application with the eIQ ExecuTorch. For details, see *Table 1*.

The eIQ ExecuTorch library is provided with a set of example applications. For details, see *Table 1*. The applications demonstrate the usage of the library in several use cases.

Name	Description	Availability
executorch_	This project contains the ExecuTorch Runtime Library source code and is used to build the ExecuTorch Runtime Library. The library is further used to build a full application using the leveraging ExecuTorch.	MIMXRT700-EVK (no camera and display support)
executorch_	Example application demonstrating the use of the ExecuTorch running a CifarNet classification model accelerated on the eIQ Neutron NPU. The Cifarnet is a small Convolutional Neural Network (CNN), trained on CIFAR-10 [1] dataset. The model clasifies the input images into 10 caterories.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG).

How to build and run executorch_cifarnet example The example needs ExecuTorch Runtime Library and Neutron Libraries.

ExecuTorch Runtime Library:

- middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a

Neutron Libraries:

- middleware/eiq/executorch/third-party/neutron/rt700/libNeutronDriver.a and
- middleware/eiq/executorch/third-party/neutron/rt700/libNeutronFirmware.a

In the example the model and the input image is already embedded into the program and ready to build and deploy to i.MX RT700, so you can continue right to the *building and deployment* section.

Convert the model and example input to C array In this section we describe where the model and example input is located in the example application sources, and how it was generated.

The **cifar10 model** ExecuTorch model is stored in boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/model_pte.h. and was generated from the cifar10_nxp_delegate.pte (see *convert_model*).

We use the xxd command to get the C array containing the model data and array size:

```
$ xxd -i cifar10_nxp_delegate.pte > model_pte_data.h
```

then use the array data and size in the model_pte.h.

As **input image** we use the image from CIFAR-10 dataset [1]. After preprocessing and normalization it is converted to bytes and located here boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/image_data.h. The preprocessing is performed as follows:

```
import torch
import torchvision
import numpy as np

batch_size = 1

transform = torchvision.transforms.Compose([
  torchvision.transforms.ToTensor(),
  torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0)

index = 0
num_images = 10
for data in test_loader:
  images, labels = data
  for image, label in zip(images,labels):
    arr = image.numpy().astype(np.float32)
    arr.tofile("img" + str(index) + "_" + str(int(label)) + ".bin")
    index = index + 1
    if index >= num_images:
      break
  if index >= num_images:
    break
```

This generates the num_images count of images from Cifar10 dataset, as input tensors for the cifar10 model and store them in corresponding .bin files. Then we use the xxd command to get the C array data and size:

```
$ xxd -i img0_3.bin > image_data_base.h
```

and again copy the array data and size in the image_data.h

Note, the img0 is the image picturing a cat, what is a class number 3.

Build, Deploy and Run

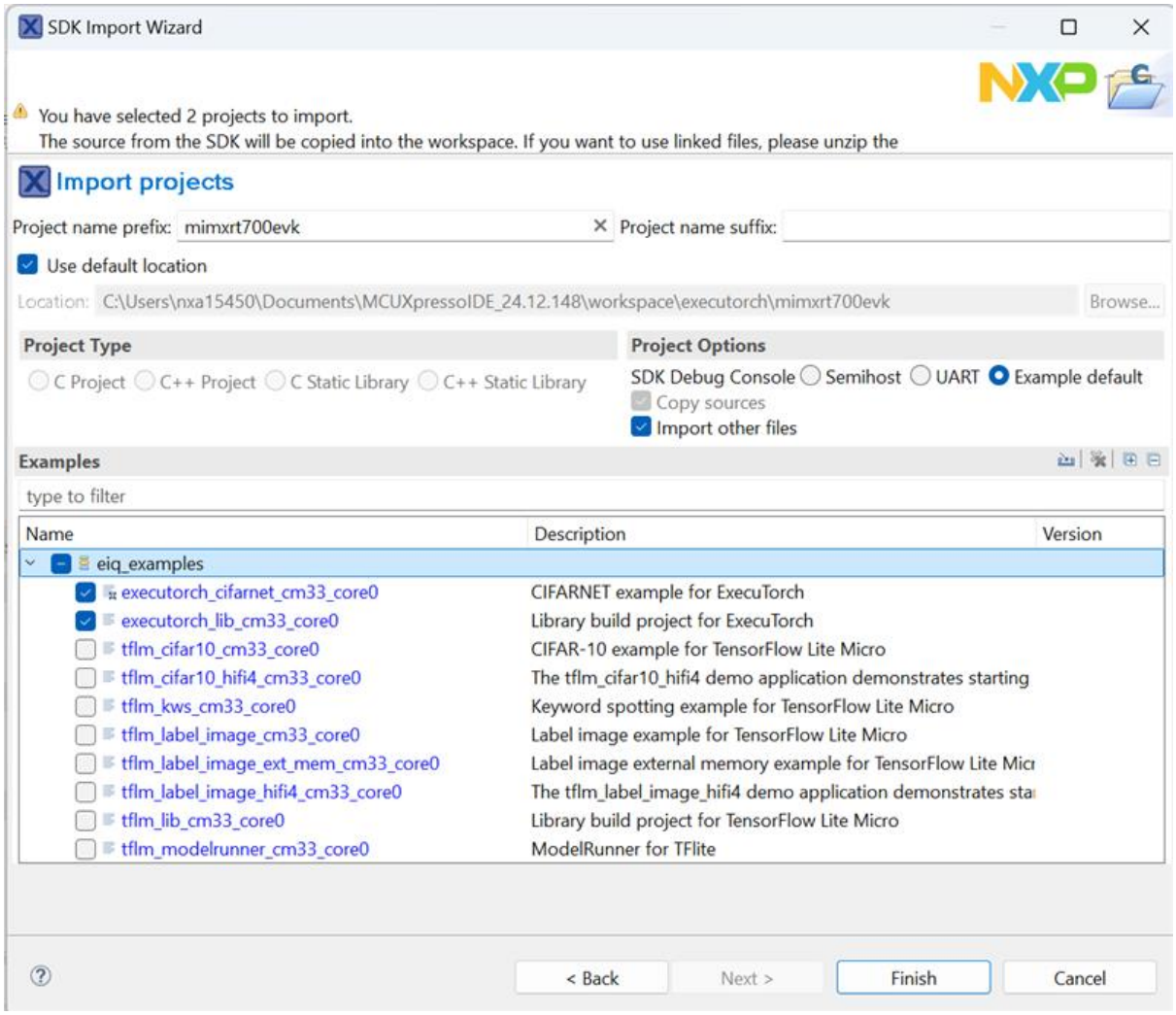
1. When using ARMGCC toolchain, the example application can be built as below. After building the example application, download it to the target with JLink as shown in *Figure 3*, an output message displays on the connected terminal as *Figure 4*.

```
$ boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/armgcc$ ./build_flash_release.sh
```

```
J-Link>loadfile C:\rt700\executorch_cifarnet_cm33_core0.elf
'loadfile': Performing implicit reset & halt of MCU.
ResetTarget() start
-- JLINK ResetTarget --
Set CM33_SYSRESETREQ_EN
CPU0 CSW: 0x03000002
ResetTarget() end - Took 88.9ms
Downloading file [C:\rt700\executorch_cifarnet_cm33_core0.elf]...
J-Link: Flash download: Bank 0 @ 0x28000000: 1 range affected (315392 bytes)
J-Link: Flash download: Total: 20.220s (Prepare: 0.145s, Compare: 2.874s, Erase: 2.558s, Program: 13.212s, Verify: 1.392s, Restore: 0
.036s)
J-Link: Flash download: Program speed: 22 KB/s
O.K.
J-Link>reset
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
ResetTarget() start
-- JLINK ResetTarget --
Set CM33_SYSRESETREQ_EN
CPU0 CSW: 0x03000002
ResetTarget() end - Took 71.5ms
J-Link>go
Memory map 'after startup completion point' is active
```

```
Model PTE file loaded. Size: 99376 bytes.
Model buffer loaded, has 1 methods
Running method forward
Setting up planned buffer 0, size 53760.
Method loaded.
Preparing inputs...
Input prepared.
Starting the model execution...
Model executed successfully.
---------------------------------------
     Inference time: 11950 us
---------------------------------------
1 outputs:
Output[0][0]: 0
Output[0][1]: 0
Output[0][2]: 0
Output[0][3]: 0.996094
Output[0][4]: 0
Output[0][5]: 0
Output[0][6]: 0
Output[0][7]: 0
Output[0][8]: 0
Output[0][9]: 0
Program complete, exiting.
```

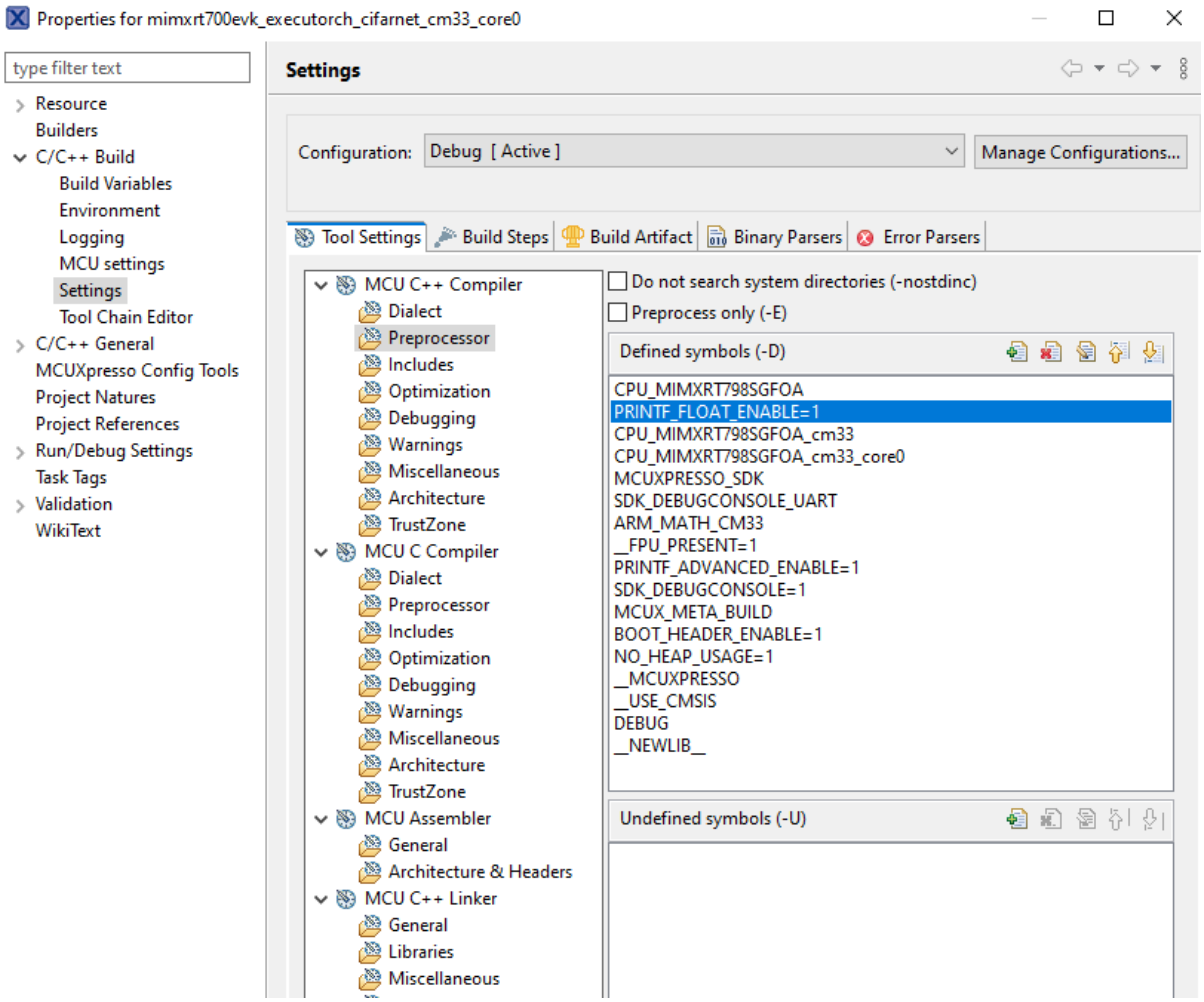
2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 5*.



After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, *Figure 6* shows the output of the executorch_cifarnet example application.

```
Model PTE file loaded. Size: 99376 bytes.
Model buffer loaded, has 1 methods
Running method forward
Setting up planned buffer 0, size 53760.
Method loaded.
Preparing inputs...
Input prepared.
Starting the model execution...
Model executed successfully.
---------------------------------------
     Inference time: 14855 us
---------------------------------------
1 outputs:
Output[0][0]: 0
Output[0][1]: 0
Output[0][2]: 0
Output[0][3]: 0.996094
Output[0][4]: 0
Output[0][5]: 0
Output[0][6]: 0
Output[0][7]: 0
Output[0][8]: 0
Output[0][9]: 0
Program complete, exiting.
```

In case of missing probabilities in the printed output, add PRINTF_FLOAT_ENABLE=1 to the Preprocessor settings for C++ and C compiler:



How to build executorch_lib **example** If you want to build a new ExecuTorch Runtime Library, follow the commands as below and use the new library to replace the default Runtime library middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a.

1. When using ARMGCC toolchain, the example application can be built as below.

```
$ boards/mimxrt700evk/eiq_examples/executorch_lib/cm33_core0/armgcc$ ./build_release.sh
$ boards/mimxrt700evk/eiq_examples/executorch_lib/cm33_core0/armgcc$ cp release/libexecutorch_lib_
↪cm33_core0.a.a ../../../../../../middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a
```

2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in the above *Figure 5*.

After building the example application, copy the new library mimxrt700evk_executorch_lib_cm33_core0\Debug\libmimxrt700evk_executorch_lib_cm33_core0.a to replace the default Runtime library mimxrt700evk_executorch_cifarnet_cm33_core0\eiq\executorch\lib\cm33\armgcc\libexecutorch.a.

[1] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009

3.3 Motor Control

3.3.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called "packet-driven BDM" interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to "packet-driven BDM", the FreeMASTER also supports a communication over [J-Link RTT]((https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to FreeMASTER community or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the src/platforms directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The “ampsdk” drivers target automotive-specific MCUs and their respective SDKs. The “dreg” implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

  The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a "middleware" component which may be downloaded along with the example applications from https://mcuxpresso.nxp.com/en/welcome.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- The official FreeMASTER middleware repository.
- Online version of this document

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.
- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.

- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample spplications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- "External" single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- "True" single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as "automatic" in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- ***src/platforms*** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- ***src/common*** folder—contains the common driver source files shared by the driver for all supported platforms. All the *.c* files must be added to the project, compiled, and linked together with the application.

- *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
- *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
- *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
- *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
- *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
- *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
- *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
- *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster_can.h* - defines the low-level message-oriented CAN API.
- *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.

  - *freemaster_net.h* - definitions related to the Network transport.
  - *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
  - *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
  - *freemaster_utils.h* - definitions related to utility code.
- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR   [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See *Driver interrupt modes*.

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR_SERIAL** - serial communication protocol
- **FMSTR_CAN** - using CAN communication
- **FMSTR_PDBDM** - using packet-driven BDM communication
- **FMSTR_NET** - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)

- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetSerialBaseAddress() to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode.
The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.
Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call ‘FMSTR_RecorderCreate()“ API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool.
Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables.
Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project.
Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions.
Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr, FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the FMSTR_Poll routine. An application interrupt can occur in the middle of the Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all **.c* files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See *Driver files* for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See *Driver configuration* for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When

a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the "idle" time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

*N * Tchar*

where:

- *N* is equal to the length of the receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance *0* which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-langiage symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type)  /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type)  /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type)  /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT*n*	Unsigned integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_SINT*n*	Signed integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_FRAC*n*	Fractional number of size *n* bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(*m*,*n*)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(*m*,*n*)	Unsigned fractional number in general UQ form (m+n total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FMSTR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")     /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")
```

(continues on next page)

(continued from previous page)

```
/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR␣
↪tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object

- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
    FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↪
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_xxx and FMSTR_PIPE_SIZE_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the nGranularity value equal to the nLength value, all data are considered as one chunk which is either written successfully as a whole or not at all. The nGranularity value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

FMSTR_TSA_TII	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	The smallest memory entity. On the vast majority of platforms, this is an unsigned 8-bit integer. On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	A single character in the communication buffer. Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

Document references

Links

- This document online: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

3.4 Wireless

3.4.1 NXP Wireless Framework and Stacks

3.4.2 EdgeFast Bluetooth

Currently we provide pdf version of those documentation, later release may convert the pdf documentation to markdown for better review and aligned format.

- EdgeFast BT PAL API Reference Manual pdf.

MCUXpressoSDK EdgeFast Bluetooth Protocol Abstraction

Introduction This document provides an overview of the EdgeFast Bluetooth Protocol Abstraction Layer stack software based on FreeRTOS OS on the NXP board with variant wireless module chipsets. This document covers hardware setup, build, and usage of the provided demo applications.

Stack API Reference EdgeFast Bluetooth Protocol Abstraction Layer is a wrapper layer on top of the bluetooth host stack. Zephyr Bluetooth host stack API is used as the basis of the EdgeFast Bluetooth Protocol Abstraction Layer with some enhancement on A2DP/SPP/HFP.

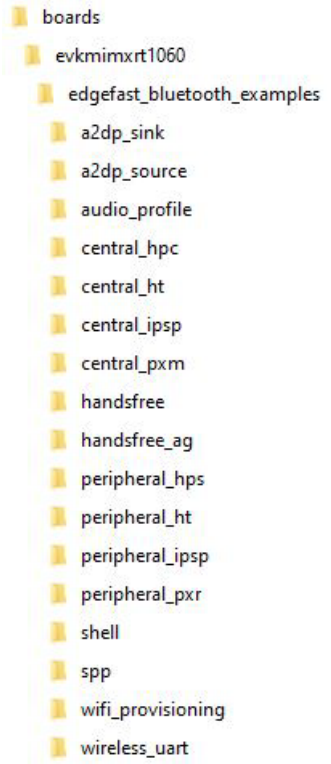
The APIs of the EdgeFast Bluetooth Protocol Abstraction Layer host stack are described in the EdgeFast Bluetooth Protocol Abstraction Layer RM document.

Note: The online document of the Zephyr Bluetooth Host stack is available here: https://docs.zephyrproject.org/latest/reference/bluetooth/index.html.

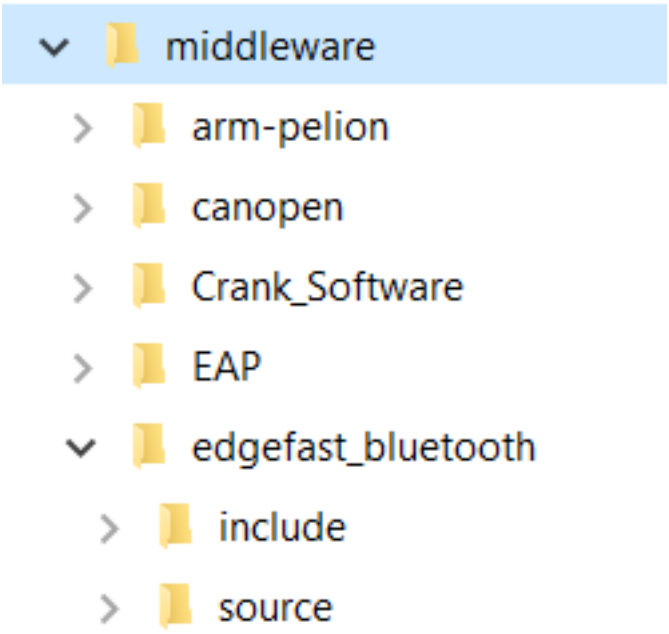
Parent topic:*Introduction*

Overview The EdgeFast Bluetooth Protocol Abstraction Layer host stack software is built based on MCUXpresso SDK. The following chapter uses RT1060 as an example, other boards have similar folder structure and corresponding document.

Folder structure The following figure shows the EdgeFast Bluetooth examples folder structure.



The following figure shows the EdgeFast Bluetooth Protocol Abstraction Layer host stack folder structure.



The following table provides information regarding the structure and description.

|**Folder**

|**Description**

| |————|————| |*boards/*

CMSIS/

devices/

docs/

middleware/

rtos/

tools/

|MCUXpresso SDK directory. Refer to Chapter 5

Release contents of MCUXpresso SDK Release Notes at *root/docs/ MCUXpresso SDK Release Notes for EVK-MIMXRT1060.pdf* to *know the details*

| |*boards/<board>/wireless/edgefast_bluetooth_examples*

|EdgeFast Bluetooth Protocol Abstraction Layer host stack example projects| |*middleware/wireless/edgefast_bluetooth*

|EdgeFast Bluetooth Protocol Abstraction Layer host stack source code

|

The EdgeFast Bluetooth folder includes two subfolders:

- **include**: This subfolder includes EdgeFast Bluetooth Protocol Abstraction Layer host stack headers.
- **source**: This subfolder includes EdgeFast Bluetooth Protocol Abstraction Layer host stack source code based on the Ethermind Bluetooth host stack APIs.

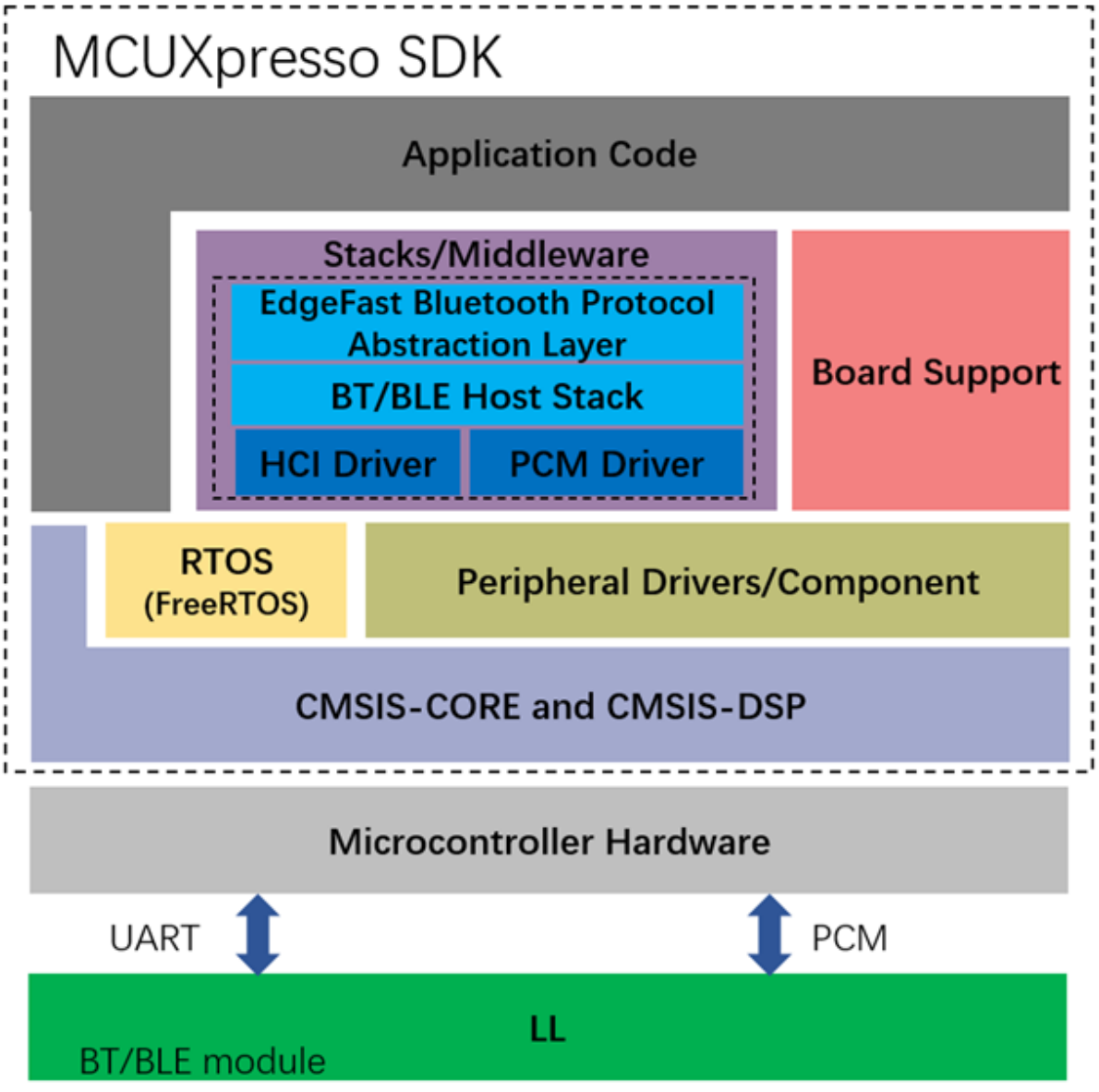
Parent topic:*Overview*

Architecture The figure Architecture of EdgeFast Bluetooth Protocol Abstraction Layer demo in MCUXpresso SDK below shows that the EdgeFast Bluetooth Protocol Abstraction Layer host stack is integrated into the MCUXpresso SDK as a middleware component. It leverages the RTOS, the board support, the peripheral driver/component, and other components in the MCUXpresso SDK. The Bluetooth application is built on top of the EdgeFast Bluetooth Protocol Abstraction Layer host stack and supports different peripheral features, Bluetooth features, and different RTOSes required by the user.

MCUXpresso SDK has the dual-chip architecture defined by EdgeFast Bluetooth Protocol Abstraction Layer project, the Bluetooth application code, and the EdgeFast Bluetooth Protocol Abstraction Layer host stack running on the reference board. For example, MIMXRT1060-EVK and the Linker Layer (LL) run on the Bluetooth modules like AW-AM457-USD, Murata Type 1XK, and Murata Type 1ZM and has single-chip architecture. Bluetooth Host stack and LL runs on the same chip, and communicate with Internal Communication Unit (IMU).

The communication between the host stack and the LL is implemented via the standard HCI UART interface and PCM interface for voice, or the IMU interface.

For details about the different components in MCUXpresso SDK, see *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf*. For details on possible hardware rework requirements, see the hardware rework guide document of the relative board. For example, Hardware Rework Guide for EdgeFast BT PAL.



Parent topic:*Overview*

Features This section provides an overview of Bluetooth features, toolchain support, and RTOS support.

Bluetooth features

- Bluetooth 5.0 compliant
- Protocol support
  - L2CAP, GAP, GATT, RFCOMM, SDP, and SM

    Note: The Enhanced Attribute (EATT) protocol is not supported in the current version. However, the support will be available in a future version.
- Classic profile
  - SPP, A2DP, and HFP
- LE profile
  - HTP, PXP, IPSP, HPS
- Integrated the Fatfs based on USB Host MSD in SDK
- Digital Audio Interface including PCM interface for HFP

Parent topic: Features

Toolchain support

- IAR Embedded Workbench for ARM®
- MCUXpresso IDE
- Keil® MDK/µVision
- Makefiles support with GCC from Arm Embedded

Note: For details on IDE Development tools version details, see Section 3, Development tools in MCUXpresso SDK Release Notes (document MCUXSDKMIMXRT106XRN). The Release Notes document is available at *root/docs/ MCUXpresso SDK Release Notes for EVK-MIMXRT1060.pdf*.

Parent topic: Features

RTOS support

- FreeRTOSTMOS

Note: The FreeRTOS static allocation feature is required by Edgefast Bluetooth. The macro configSUPPORT_STATIC_ALLOCATION needs to be set to enable this feature.

Parent topic: Features

Parent topic:*Overview*

Examples list

- The following examples are provided. Not all the examples are implemented on all the boards. See the board package for a list of the implemented examples.

- **central_hpc (central http proxy service client)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for HPS Server and programs a set of characteristics that configures a Hyper Text Transfer Protocol (HTTP) request, initiates request, and reads the response once connected.
- **central_ht (central health thermometer)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for health thermometer sensor and reports the die temperature readings once connected.
- **central_ipsp (central Internet protocol support profile)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes connection to the peripheral with the strongest signal. The application specifically looks for IPSP Service and communicates between the devices that support IPSP. Once connected, the communication is done using IPv6 packets over the Bluetooth Low Energy transport.
- **central_pxm (central proximity monitor)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for Proximity Reporter.
- **peripheral beacon**: Demonstrates the Bluetooth Low Energy Peripheral role, This application implements types of beacon applications.
  - **beacon**: Demonstrates the Bluetooth Low Energy Broadcaster role functionality by advertising Company Identifier, Beacon Identifier, UUID, A, B, C, RSSI.
  - **Eddystone**: The Eddystone Configuration Service runs as a GATT service on the beacon while it is connectable and allows configuration of the advertised data, the broadcast power levels, and the advertising intervals.
  - **iBeacon**: Demonstrates the Bluetooth Low Energy Broadcaster role functionality by advertising an Apple iBeacon.
- **peripheral_hps (peripheral http proxy service)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the HTTP Proxy GATT Service.
- **peripheral_ht (peripheral health thermometer)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the HT (Health Thermometer) GATT Service. Once a device connects, it generates dummy temperature values.
- **peripheral_ipsp (peripheral Internet protocol support profile)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the Internet Protocol Support GATT Service.
- **peripheral_pxr (peripheral proximity reporter)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the Proximity Reporter (including LLS, IAS, and TPS) GATT Service.
- **wireless uart**: The application automatically starts advertising the wireless uart service and connects to the wireless uart service after the role switch. The wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.
- **spp (serial prot profile)**: Application demonstrates the use of the SPP feature.
- **handsfree**: Application demonstrating usage of the Hands-free Profile (HFP) feature.

- **handsfree_ag**: Application demonstrating usage of the Hands-free Profile Audio Gateway (HFP-AG) feature.
- **a2dp_sink**: Application demonstrating how to use the a2dp sink feature.
- **a2dp_source**: Application demonstrating how to use the a2dp source feature.
- **audio_profile**: Demonstrates the following functions.
  * There are five parts working in the demo: AWS cloud, Android app, audio demo (running on RT1060), U-disk, and Bluetooth headset.
  * With an app running on the smartphone (Android phone), the end user connects to the AWS cloud and controls the audio demo running on the RT1060 EVK board through AWS cloud. Some operations like play, play next, and pause are used to control the media play functionalities.
  * Audio demo running on the RT1060 EVK board connects to the AWS through WiFi. A connection establishes between the RT1060 EVK board and a Bluetooth headset. To get the media resource (mp3 files) from the U-disk, an HS USB host is enabled, and a U-disk with mp3 files is connected to RT1060 EVK board via the USB port. The audio demo searches the root directory of the U-disk for the music files (only mp3 files are supported) and uploads the song file list to AWS. The song list is shown in the app running on the smartphone. The music can then be played out via the Bluetooth headset once end user controls the app to play the mp3 file.
- **wifi_provisioning**: Demonstrates the WiFi provisioning service that safely sends credential from phone to device over Bluetooth low energy. By default, AWS Wi-Fi provisioning demo starts advertising if the Wi-Fi access point (AP) is not configured and waits for the Wi-Fi AP configuration. After connecting to the Android APK, the demo executes the request from cellphone and sends the response. When the Wi-Fi AP is configured, the Shadow demo connects to the AWS via Wi-Fi and publishes the configured Wi-Fi AP information.
- **shell**: Shell application demonstrating the shell mode of the simplified Adapter APIs.

Parent topic:*Overview*

Hardware For dual-chip implementation, the Bluetooth demo runs on a (reference board) along with the ported EdgeFast Bluetooth Protocol Abstraction Layer API host stack. The Linker Layer (LL) runs on a wireless module. A standard UART HCI and PCM is used to communicate between the two boards, the IMU is used to communicate in between. The Bluetooth host and controller stack run on different boards. The demo hardware requires two different boards; a development board for host stack and application and a wireless module adapter board for controller running. For example, the evkmimxrt1060 and uSD-15x15 Adapter Board for AW-AM457-uSD board, or any of the supported Murata modules with the Murata uSD-M.2 adapter. For details on the board hardware requirement and board setting, see the following documents. For one-chip implementation, the Bluetooth demo, EdgeFast Bluetooth Protocol Abstraction Layer API host stack, and LL run on one chip and they communicate with IMU.

- Hardware rework guide document of the relative board, Hardware Rework Guide for MIMXRT1060-EVK and AW-AM457-uSD, or Hardware Interconnection Guide for i.MX RT EVKs and Murata M.2 modules.
- Readme file of the examples.

Reference boards list

- MIMXRT1170: For details, see the quick start guide of this reference board (MIMXRT1170).
- MIMXRT685-EVK: For details, see the quick start guide of this reference board (MIMXRT685-EVK).

- MIMXRT595-EVK: For details, see the quick start guide of this reference board. (MIMXRT595-EVK).
- MIMXRT1050-EVKB: For details, see the quick start guide of this reference board (MIMXRT1050-EVKB).

Parent topic:*Hardware*

Dual-chip wireless module list

Module	HCI
uSD-15x15 Adapter Board for AW-AM457-uSD	UART
uSD-15x15 Adapter Board for AW-CM358-uSD	UART
uSD-15x15 Adapter Board for AW-AM510-uSD	UART
AW-CM358MA	UART
AW-CM510MA	UART
K32W061	UART
Murata uSD-M.2 Adapter (LBEE0ZZ1WE-uSD-M2) and Embedded Artists 1ZM M.2 Module (EAR00364)	UART
Murata uSD-M.2 Adapter (LBEE0ZZ1WE-uSD-M2) and Embedded Artists 1XK M.2 Module (EAR00385)	UART

For details on AzureWave module, see the quick start guide of this reference board AW-AM457-uSD, AW-CM358-uSD, AW-CM358MA, AW-AM510-uSD, AW-CM510MA, and K32W061.

For Murata documentation, refer to the Quick Start Guide and User Guide here.

Note: The boards and wireless module lists are not random combination. For the wireless module support list of specific board, see the readme.txt of each example.

Parent topic:*Hardware*

Demo This topic lists the steps to run a demo application using IAR, steps to run a demo application using MCUXpresso IDE, and steps to download LL firmware from the reference board. The following chapter uses RT1060 and peripheral_ht as an example.

Before you run the example, see the readme.txt in current the peripheral_ht directory and the Hardware Rework Guide for EdgeFast BT PAL document to set the jumper and connect the wireless module with development board.

The uSD type wireless module is similar to the Development board connector in the Run an IAR example section. If the module is M2 type, connect the module to the onboard M2 interface.

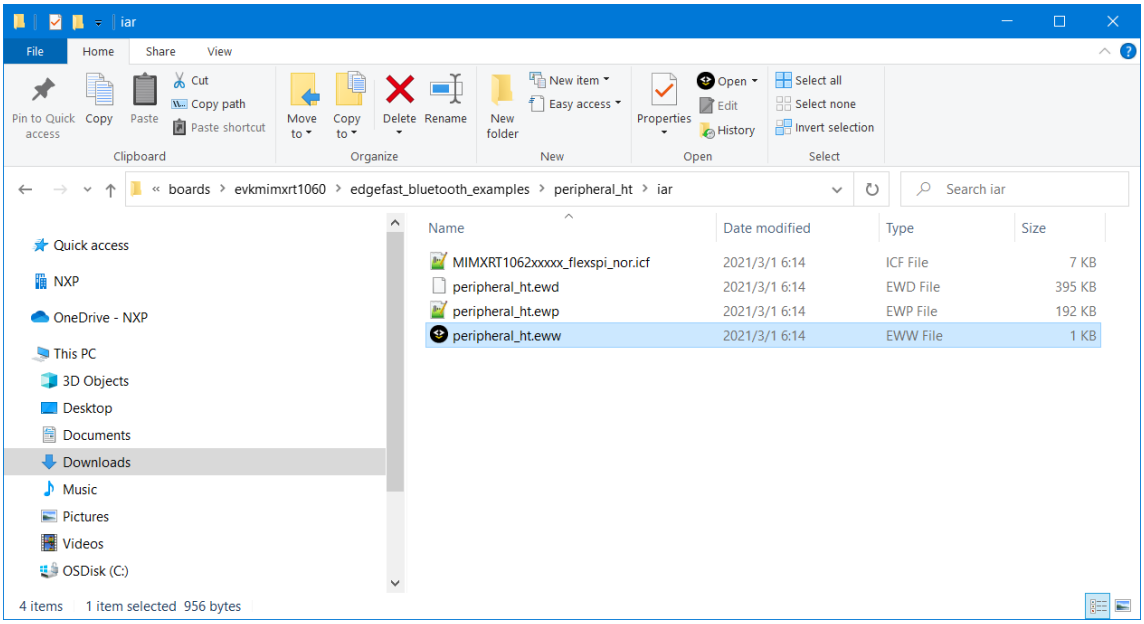
Run a demo application using IAR This document uses EVKRT1060 EdgeFast Bluetooth Protocol Abstraction Layer API example to describe the steps to open a project, build an example, and run a project. For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide*(document MCUXSDKGSUG) at*root/docs/Getting Started with MCUXpresso SDK.pdf*.

Open an IAR example For the IAR Embedded Workbench, unpack the contents of the archive to a folder on a local drive.

1. The example projects are available at:

   <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/iar

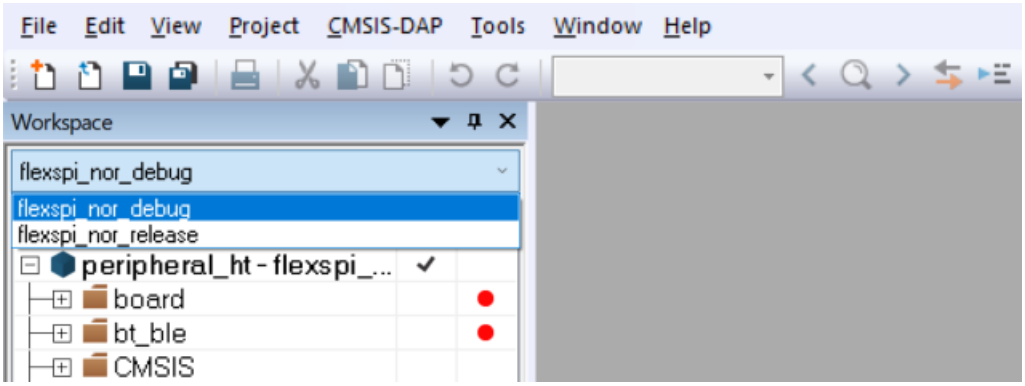
2. Open the IAR workspace file. For example, the highlighted *.eww format file



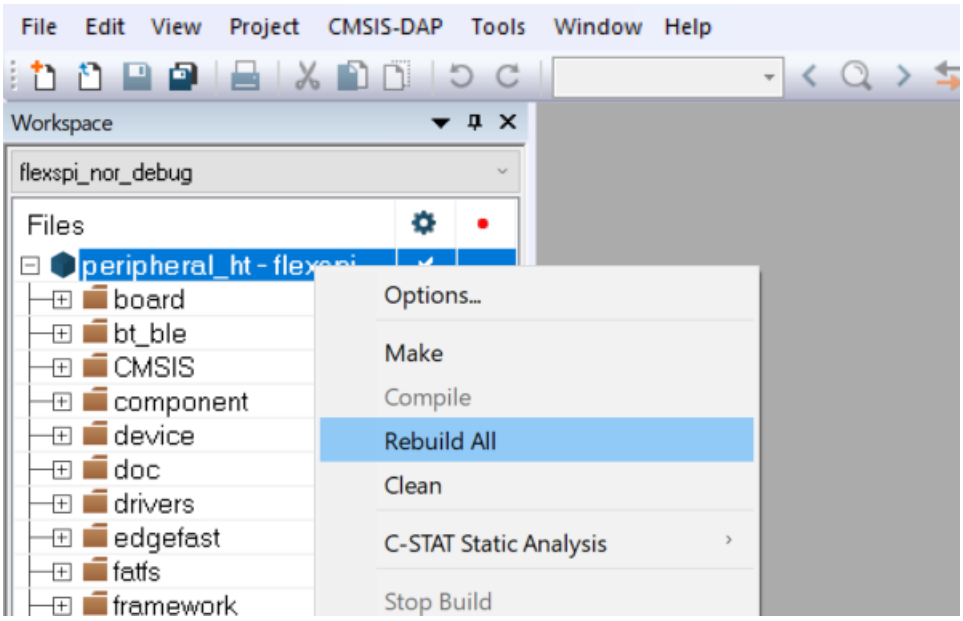
Parent topic: Run a demo application using IAR

Build an IAR example

1. Select flexspi_nor_debug or flexspi_nor_release configurations from the drop-down selector above the project tree in the workspace.



2. Build the EdgeFast Bluetooth Protocol Abstraction Layer project.

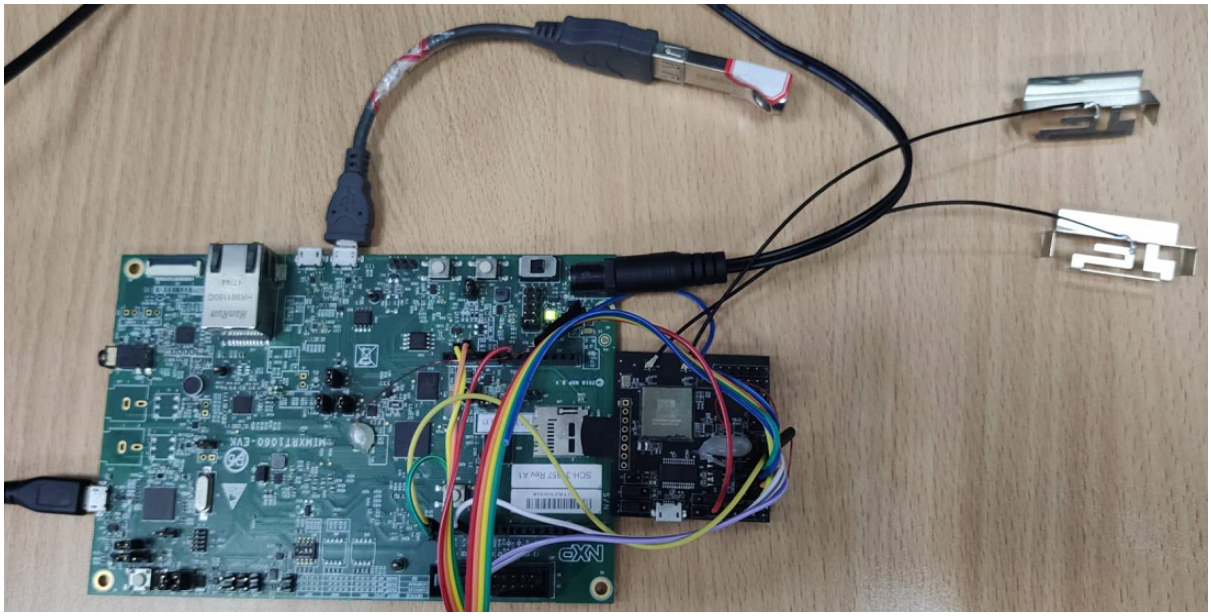


Note: Wireless module does not have flash hardware and requires 512 KB image loaded from board (such as RT1060) on system startup. The 512 KB image is kept on RT1060 side and only flexspi_nor target is supported for Bluetooth examples. Other targets are not supported because memory size limit.

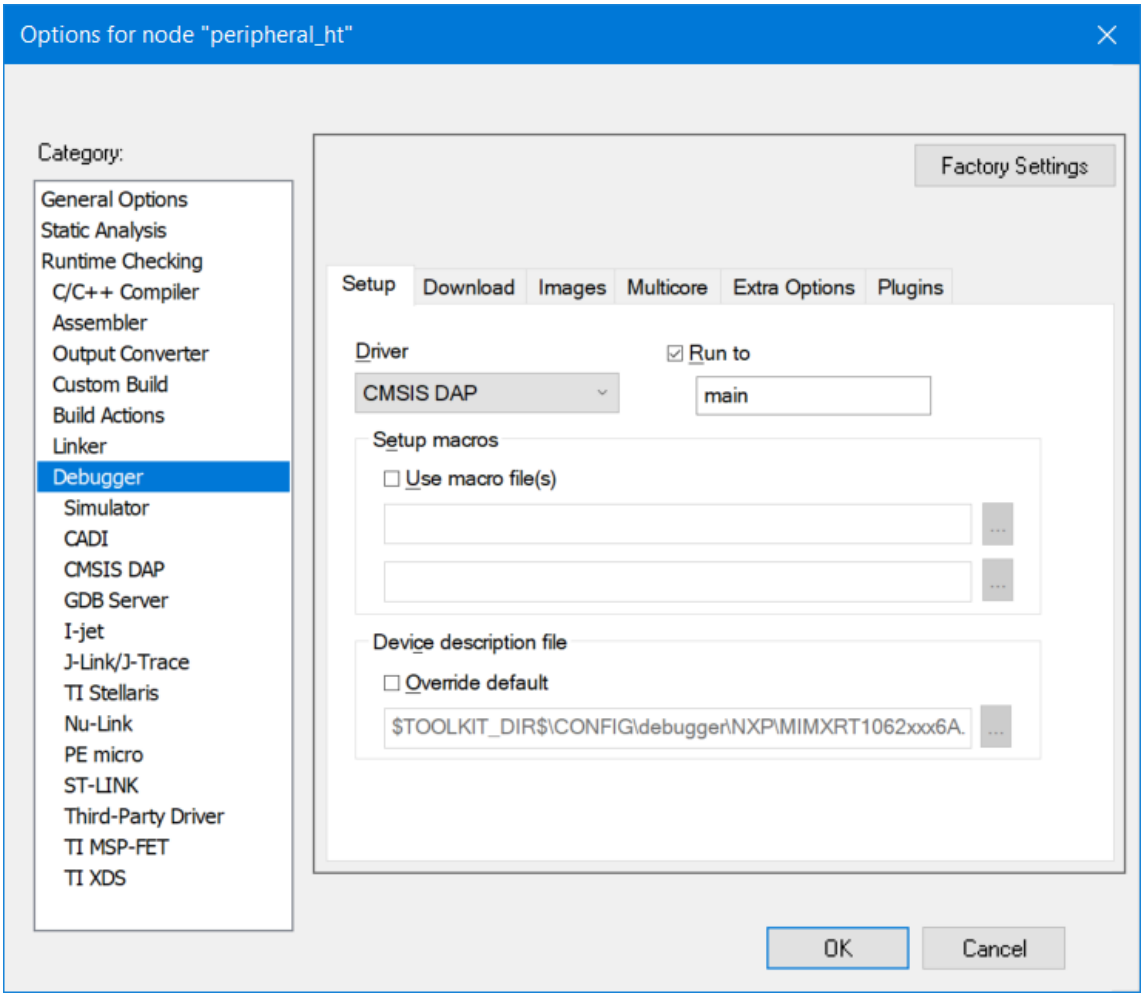
Parent topic: Run a demo application using IAR

Run an IAR example This document uses the peripheral_ht as an example to describe the steps to run an example. For details on other projects and compilers, see the readme file in the corresponding example directory.

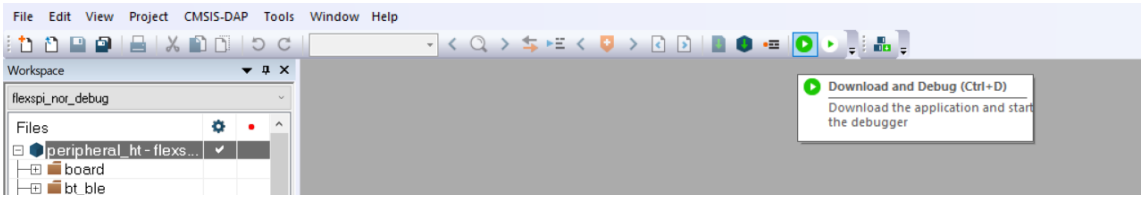
The following figure shows the connection of RT1060 and the uSD wireless module.



1. Connect the USB debug console port to PC. For example, connect J14 of EVKRT1060 to the PC.
2. Connect a 5 V power source to the J1 jack in the Wireless module board.
3. Make the appropriate debugger settings in the project options window, as shown in the figure below.



4. Click the **Download and Debug** button to flash the executable onto the board, as shown in the following figure. After the download is complete, if you must test the function of HFP, stop IAR debugging, and then connect the PCM interface. Reset the target board by manually.



5. Linker layer (LL) Firmware running in wireless module loads from EVKRT1060 by SDIO interface, so need take a bit time to download the LL firmware, “Initialize AW-AM457-uSD Driver” prints in the debug console. For example, it depends on the firmware. For details, see readme.txt.

Note: The projects are configured to use “CMSIS DAP” as the default debugger. Ensure that the OpenSDA chip of the board contains a CMSIS. DAP firmware or that the debugger selection corresponds to the physical interface used to interface to the board.

Parent topic: Run a demo application using IAR

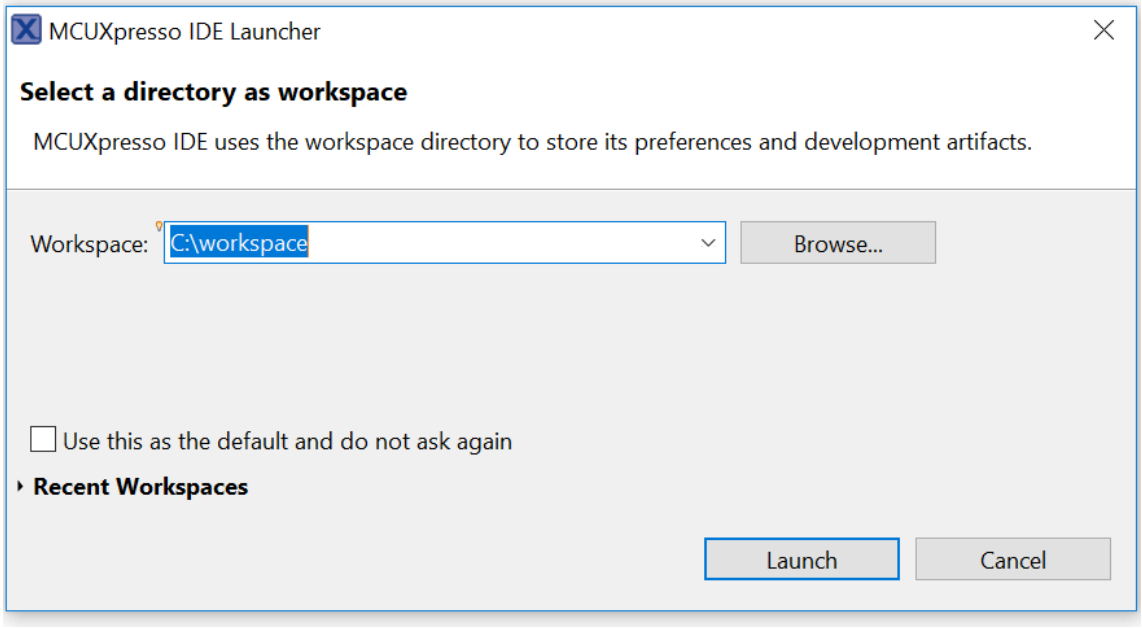
Parent topic:*Demo*

Run a demo application using MCUXpresso IDE This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MCUXpresso IDE.

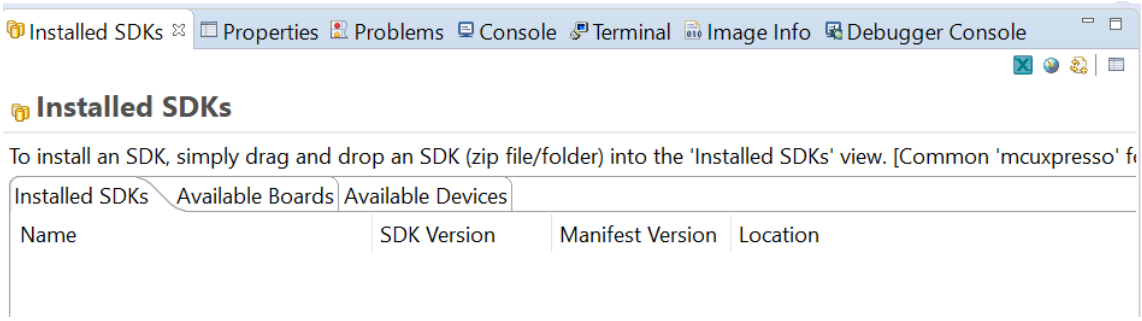
For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf* and refer to the readme file in the corresponding demo's directory.

Open an MCUXpresso IDE example

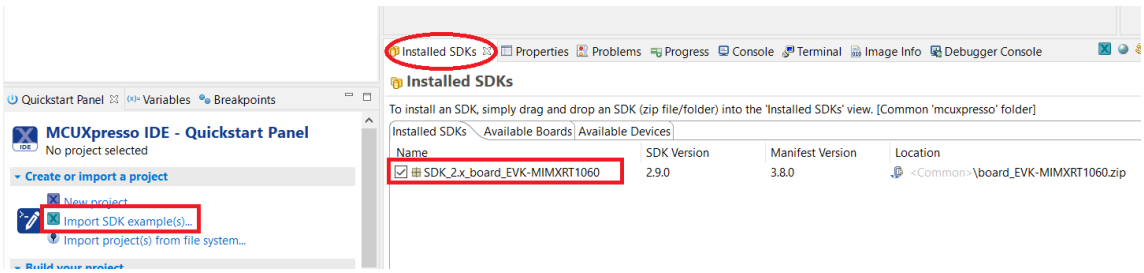
1. Open MCUXpresso IDE and open an existing or a new workspace location.



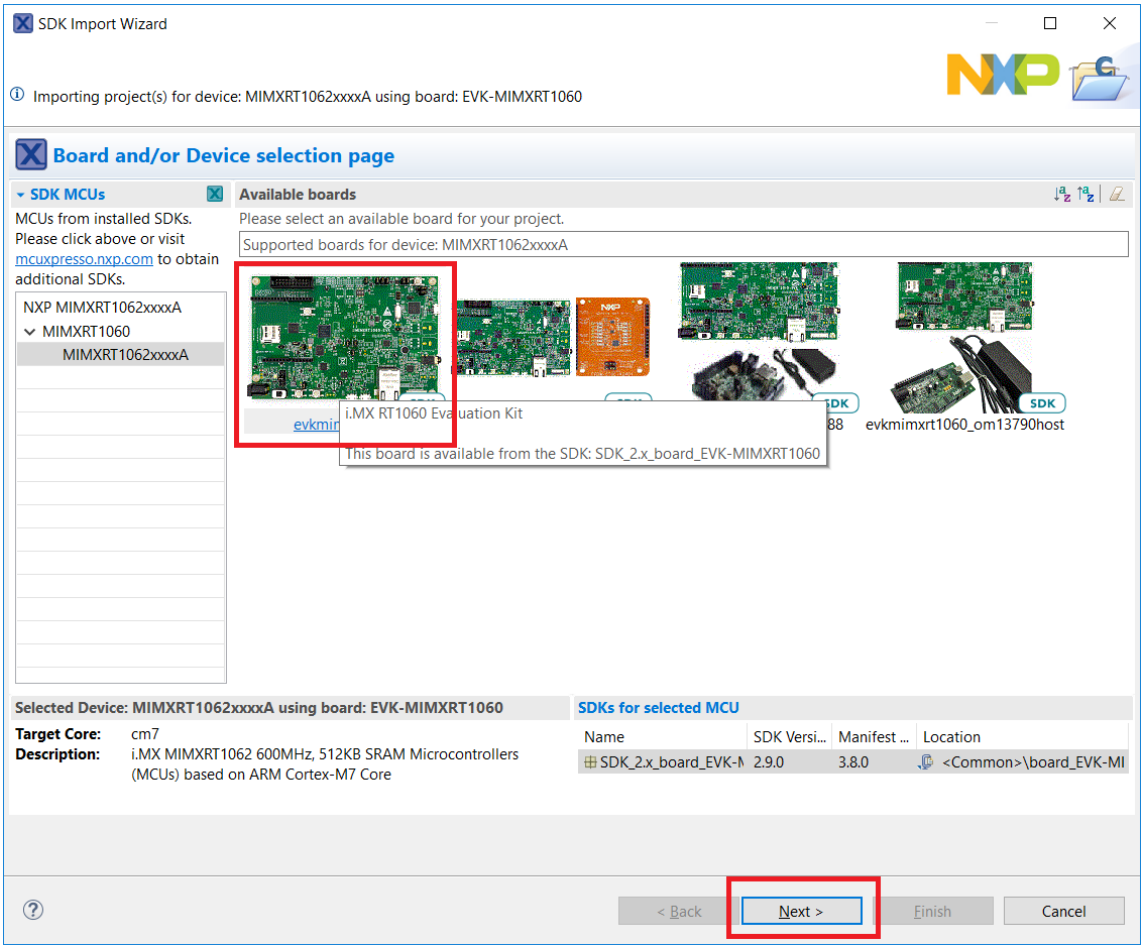
2. Drag and drop the package archive into the MCUXpresso Installed SDKs area in the lower right of the main window.



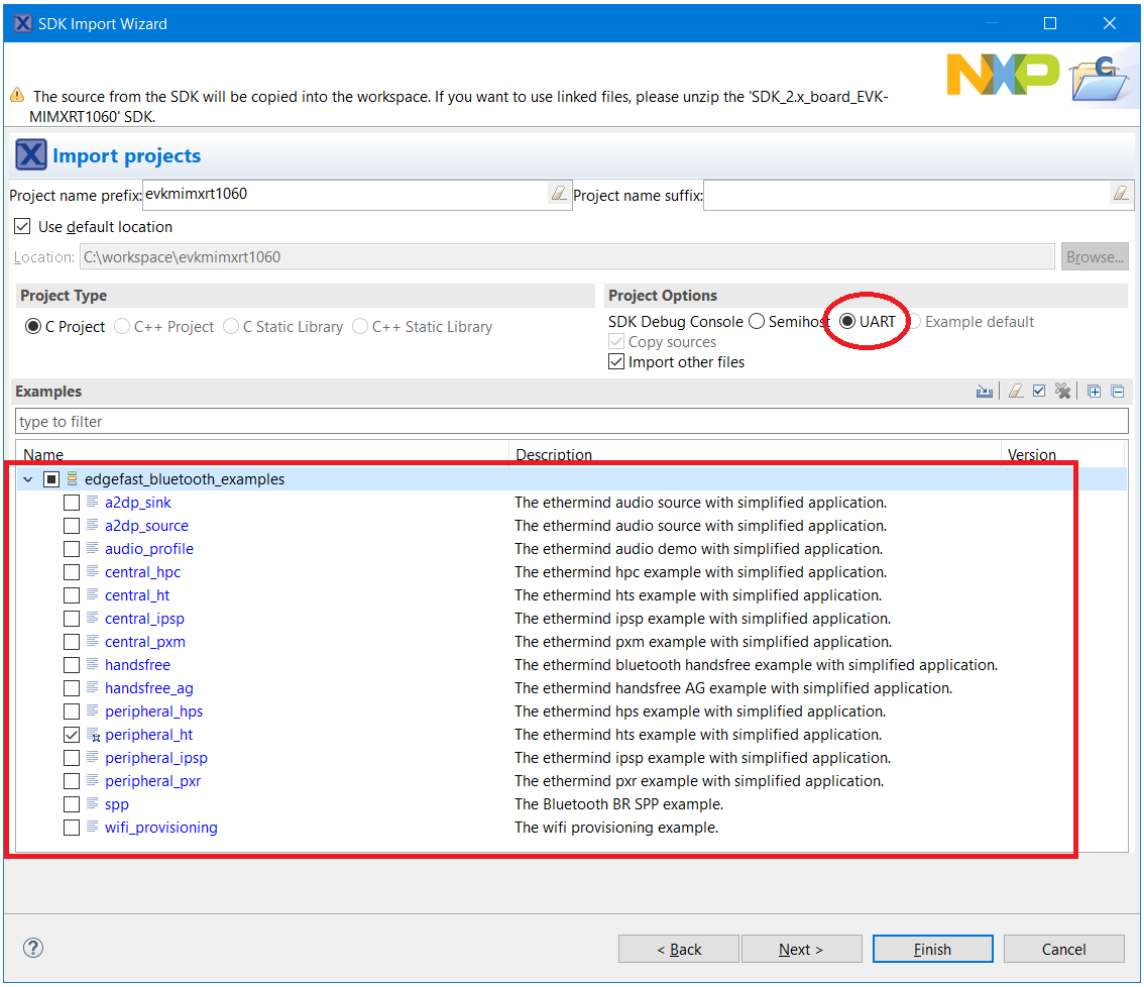
3. After the SDK is loaded successfully, select the **Import the SDK examples(s)...** to add examples to your workspace.



4. Select the evkmimxrt1060 board and click the **Next** button to select the desired example(s).



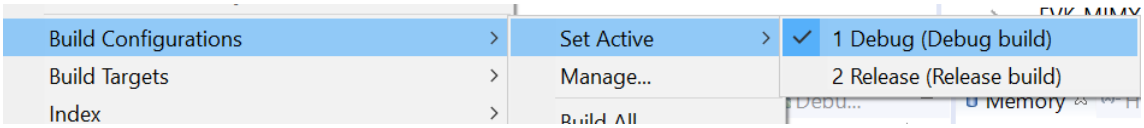
5. Select the evkmimxrt1060 board EdgeFast Bluetooth example. For example, peripheral_ht.
6. Ensure to change SDK debug console from **Semihost** to **UART**.
7. Click **Finish**.



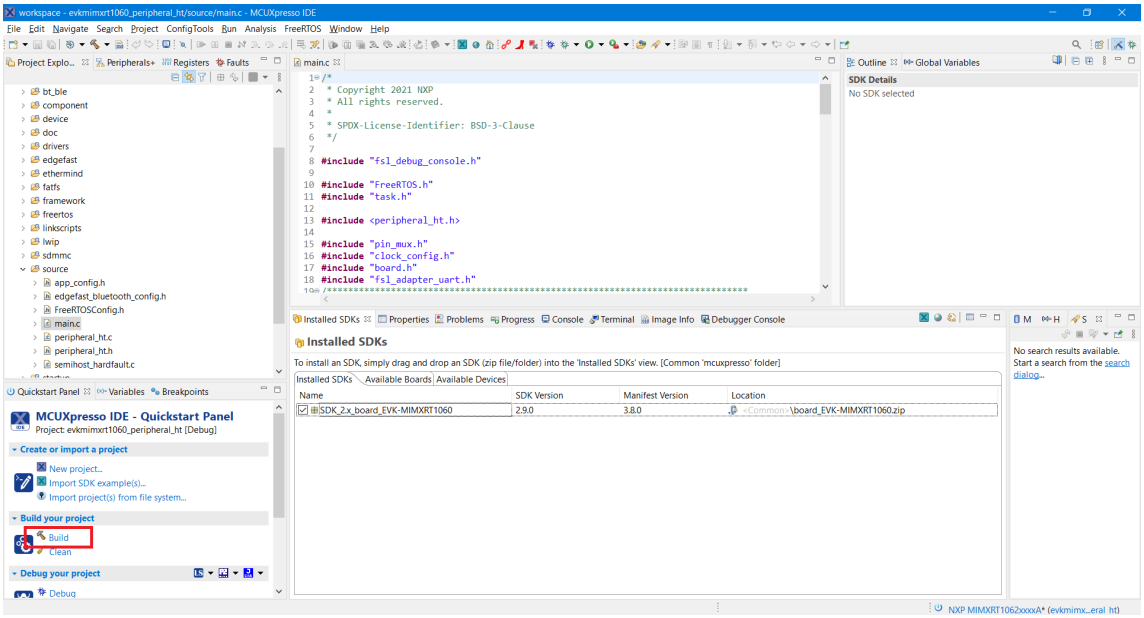
Parent topic: Run a demo application using MCUXpresso IDE

Build an MCUXpresso IDE example

1. Select desired target for your project.



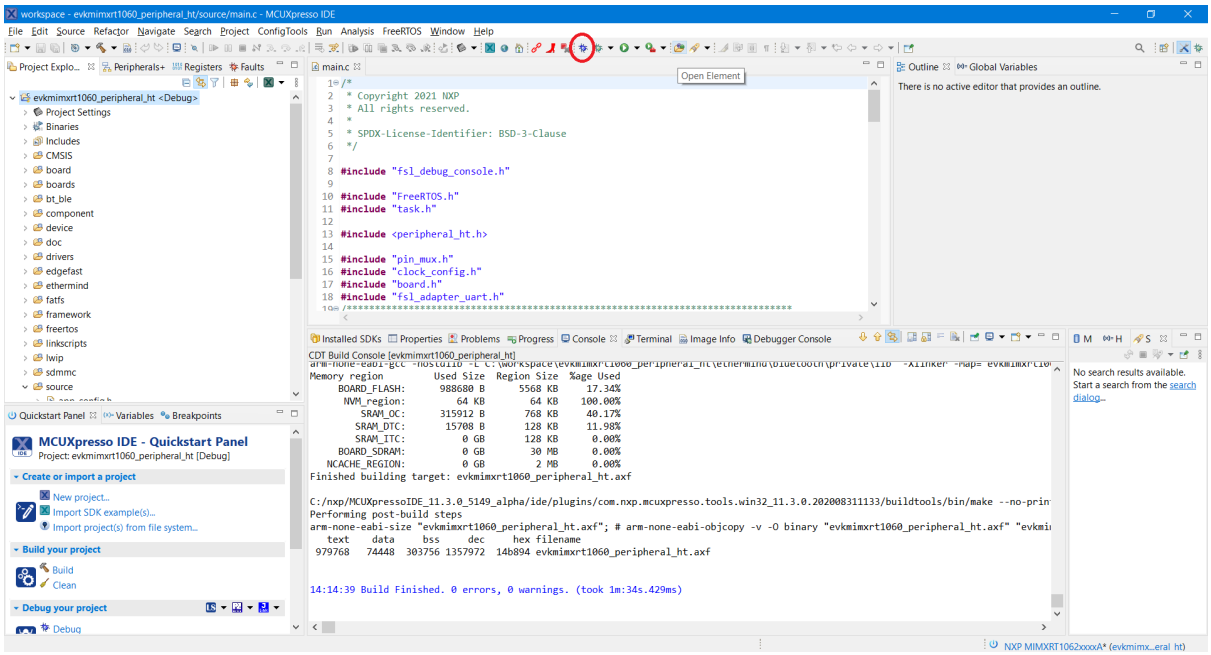
2. Build MCUXpresso IDE EdgeFast Bluetooth Protocol Abstraction Layer project.



Parent topic: Run a demo application using MCUXpresso IDE

Run an MCUXpresso IDE example For MCUXpresso IDE project running, all steps are similar to Run an IAR example except the steps of downloading image from compiler.

To download MCUXpresso IDE image to board, click the **Debug** button to download the executable file onto the board.



Parent topic: Run a demo application using MCUXpresso IDE

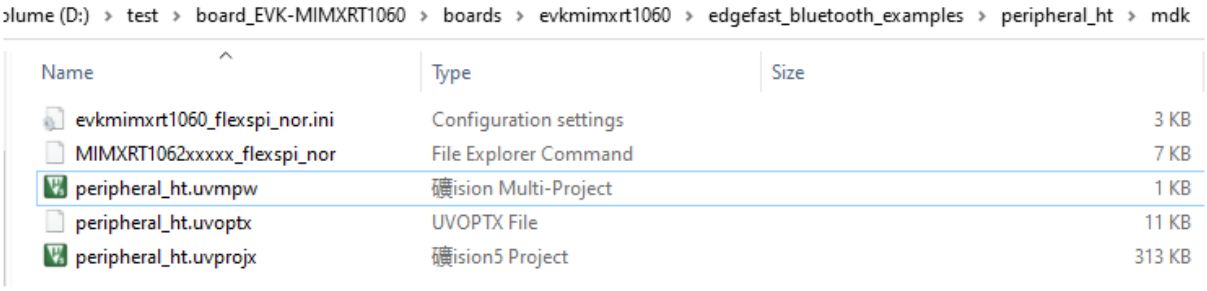
Parent topic:*Demo*

Run a demo application using MDK This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MDK.

For details, see the related section in the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG) in the directory *root/docs/* and the readme file in the corresponding demo's directory.

Open an MDK project For the IAR Embedded Workbench, unpack the contents of the archive to a folder on a local drive.

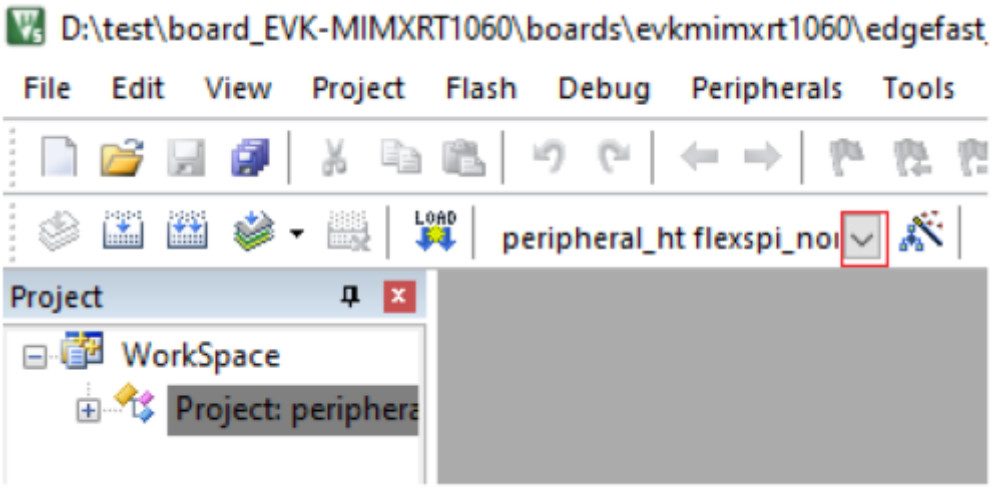
1. The example projects are available at: <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/mdk.
2. Open the mdk workspace file. For example, the highlighted *.uvmpw format file.



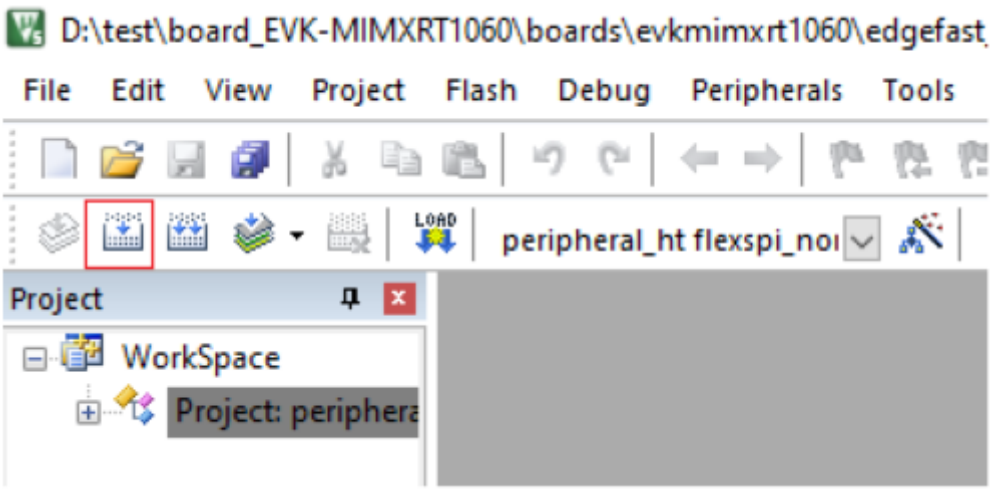
Parent topic: Run a demo application using MDK

Build an MDK example To build an MDK example:

1. Select *flexspi_nor_debug* or *flexspi_nor_release* configurations from the drop-down selector above the project tree in the workspace.



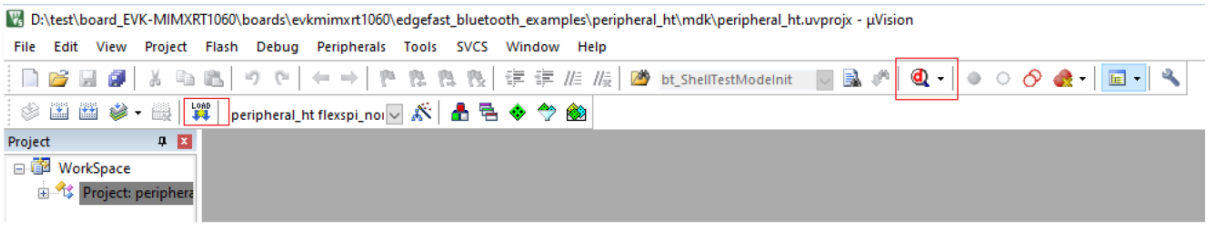
2. Click the highlighted icon to build the EdgeFast Bluetooth Protocol Abstraction Layer project.



Parent topic: Run a demo application using MDK

Run an MDK example For MDK project running, all steps are similar to Run an IAR example except the steps of downloading image from compiler.

To download the MDK image to the board, click the **Debug** button. The executable file downloads to the board.



Parent topic: Run a demo application using MDK

Parent topic:*Demo*

Run a demo application using Arm GCC This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MDK.

For details, see the related section in *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG) at *root/docs/* and the readme file in the corresponding demo's directory.

Setup tool chains See the section "Run a demo using Arm GCC" of getting start document. For example, *Getting Started with MCUXpresso SDK for MIMXRT1160-EVK*.

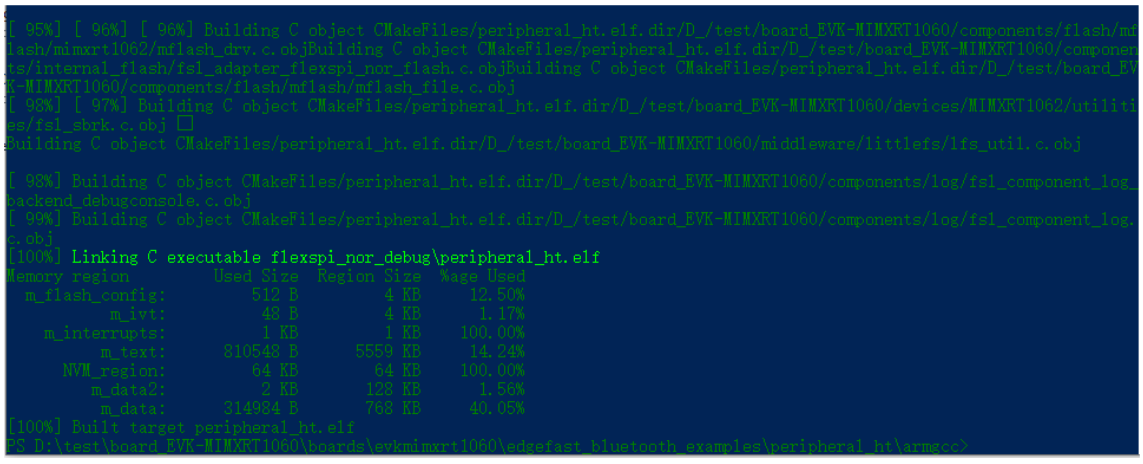
Parent topic: Run a demo application using Arm GCC

Build a GCC example To build a GCC example:

1. Change the directory to the project directory: <install_dir>\boards\evkmimxrt1060\edgefast_bluetooth_examples\peripheral_ht\armgcc.
2. Run the build script.

   For windows, the script is build_flexspi_nor_debug.bat/ build_flexspi_nor_release.bat.

   The build output is shown in the following figure.



Parent topic: Run a demo application using Arm GCC

Run a GCC example Refer to the section "Run a demo using Arm GCC" of the getting start document. For example, see Getting Started with MCUXpresso SDK for MIMXRT1060-EVK. The peripheral_ht.elf is the target to download.

Parent topic: Run a demo application using Arm GCC

Parent topic:*Demo*

Download Linker Layer firmware from the reference board Download the Linker Layer (LL) Firmware from Reference board EVKRT1060 by SDIO interface before running the Bluetooth Controller stack. The LL download is necessary because wireless module does not support flash.

Parent topic:*Demo*

Change board-specific parameters There are some board-specific parameters that can be changed in the application layer for EdgeFast BT PAL.

Change HCI UART parameters Since the controller can support different baud rates, the demo provides an interface with configurable baud rates. The function*controller_hci_uart_get_configuration* is used to get HCI UART parameters, including the instance, default baud rate, which depends on the controller, running baud rate which defined by macro BOARD_BT_UART_BAUDRATE and so on. If this function returns '0' and the running baud rate is inconsistent with the default baud rate, EdgeFast BT PAL switches the baud rate of the controller to the running baud rate.

Parent topic: Change board-specific parameters

Change USB Host stack parameters Since the board supports multiple USB ports, the demo provides a configurable interface for USB Host stack. The function*USB_HostGetConfiguration* received the instance of USB for EdgeFast BT PAL. For the case where there is a USBPHY, the demo configures the properties of the PHY through*USB_HostPhyGetConfiguration*.

Note: There are series of hex bytes printed on the console after the wireless module resets. However, it does not impact the EdgeFast BT PAL application running.

Parent topic: Change board-specific parameters

Parent topic:*Demo*

Known issues This section provides a list of known issues in the release package.

Notes This section provides a list of notes to use EdgeFast Bluetooth stack

- the follow configuration items related to resource needs more attention
  - CONFIG_BT_MAX_CONN The max connections that can be created.
  - CONFIG_BT_MAX_PAIRED The max supported paired devices.
  - CONFIG_BT_BUF_EVT_RX_COUNT The max received hci events and acl data packets at one time if the sys work queue task is blocked. One example is: when LE connection is created and HCI_LE_Enhanced_Connection_Complete is received, the sys work queue task is busy with processing the HCI_LE_Enhanced_Connection_Complete. If the received hci events exceed CONFIG_BT_BUF_EVT_RX_COUNT, it may leads potential issue, please increase value of the macro.
- All the EdgeFast Bluetooth API should be called only after EdgeFast Bluetooth is initialized.
- Don't send HCI cmd from the sys work queue task or any stack's callbacks.

EdgeFast BT PAL configuration documentation **CONFIG_BT_BUF_RESERVE**

Buffer reserved length, suggested value is 8.

CONFIG_BT_SNOOP

Whether enable bt snoop feature, 0 - disable, 1 - enable.

CONFIG_BT_HCI_CMD_COUNT

Number of HCI command buffers, ranging from 2 to 64. Number of buffers available for HCI commands Range 2 to 64 is valid.

CONFIG_BT_RX_BUF_COUNT

Number of HCI RX buffers, ranging from 2 to 255. Number of buffers available for incoming ACL packets or HCI events from the controller Range 2 to 255 is valid.

CONFIG_BT_RX_BUF_LEN

Maximum supported HCI RX buffer length, ranging from 73 to 2000. Maximum data size for each HCI RX buffer. This size includes everything starting with the ACL or HCI event headers. Note that buffer sizes are always rounded up to the nearest multiple of 4, so if this Kconfig value is something else then there is some wasted space. The minimum of 73 has been taken for LE SC which has an L2CAP MTU of 65 bytes. On top of this, The L2CAP header (4 bytes) and the ACL header (also 4 bytes) which yields 73 bytes. Range is 73 to 2000.

CONFIG_BT_HCI_RESERVE

Reserve buffer size for user. Headroom that the driver needs for sending and receiving buffers. Add a new 'default' entry for each new driver.

CONFIG_BT_DISCARDABLE_BUF_COUNT

Number of discardable event buffers, if the macro is set to 0, disable this feature, if greater than 0, this feature is enabled. Number of buffers in a separate buffer pool for events which the HCI driver considers discardable. Examples of such events could be , for example, Advertising Reports. The benefit of having such a pool means that if there is a heavy inflow of such events it does not cause the allocation for other critical events to block and may even eliminate deadlocks in some cases.

CONFIG_BT_DISCARDABLE_BUF_SIZE

Size of discardable event buffers, ranging from 45 to 257. Size of buffers in the separate discardable event buffer pool. The minimum size is set based on the Advertising Report. Setting the buffer can save memory if with size set differently from that of the CONFIG_BT_RX_BUF_LEN. range is 45 to 257.

CONFIG_BT_HCI_TX_STACK_SIZE

HCI TX task stack size needed for executing bt_send with specified driver, should be no less than 512.

CONFIG_BT_HCI_TX_PRIO

HCI TX task priority.

CONFIG_BT_RX_STACK_SIZE

Size of the receiving thread stack. This is the context from which all event callbacks to the application occur. The default value is sufficient for basic operation, but if the application needs to do advanced things in its callbacks that require extra stack space, this value can be increased to accommodate for that.

CONFIG_BT_RX_PRIO

RX task priority.

CONFIG_BT_PERIPHERAL

Peripheral Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Peripheral role support.

CONFIG_BT_BROADCASTER

Broadcaster Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Broadcaster role support.

CONFIG_BT_EXT_ADV

Extended Advertising and Scanning support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this to enable Extended Advertising API support. This enables support for advertising with multiple advertising sets, extended advertising data, and advertising on LE Coded PHY. It enables support for receiving extended advertising data as a scanner, including support for advertising data over the LE coded PHY. It enables establishing connections over LE Coded PHY.

CONFIG_BT_CENTRAL

Central Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Central role support.

CONFIG_BT_WHITELIST

Enable whitelist support. This option enables the whitelist API. This takes advantage of the whitelisting feature of a Bluetooth LE controller. The whitelist is a global list and the same whitelist is used by both scanner and advertiser. The whitelist cannot be modified while it is in use. An Advertiser can whitelist which peers can connect or request scan response data. A scanner can whitelist advertiser for which it generates advertising reports. Connections can be established automatically for whitelisted peers.

This option deprecates the bt_le_set_auto_conn API in favor of the bt_conn_create_aute_le API.

CONFIG_BT_DEVICE_NAME

Bluetooth device name. Name can be up to 248 bytes long (excluding NULL termination). Can be empty string.

CONFIG_BT_DEVICE_APPEARANCE

Bluetooth device appearance. For the list of possible values, see the link: www.bluetooth.com/specifications/assigned-numbers.

CONFIG_BT_DEVICE_NAME_DYNAMIC

Allow to set Bluetooth device name on runtime. Enabling this option allows for runtime configuration of Bluetooth device name.

CONFIG_BT_ID_MAX

Maximum number of local identities, range 1 to 10 is valid. Maximum number of supported local identity addresses. For most products, this is safe to leave as the default value (1). Range 1 to 10 is valid.

CONFIG_BT_CONN

Connection enablement, if the macro is set to 0, feature is disabled, if 1, feature is enabled.

CONFIG_BT_MAX_CONN

it is the max connection supported by host stack. Maximum number of simultaneous Bluetooth connections supported.

CONFIG_BT_HCI_ACL_FLOW_CONTROL

Controller to host ACL flow control support. Enable support for throttling ACL buffers from the controller to the host. This is useful when the host and controller are on separate cores, since it ensures that we do not run out of incoming ACL buffers.

CONFIG_BT_PHY_UPDATE

PHY Update, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable support for Bluetooth 5.0 PHY Update Procedure.

CONFIG_BT_DATA_LEN_UPDATE

Data Length Update. If the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable support for Bluetooth v4.2 LE Data Length Update procedure.

CONFIG_BT_CREATE_CONN_TIMEOUT

Timeout for pending LE Create Connection command in seconds.

CONFIG_BT_CONN_PARAM_UPDATE_TIMEOUT

Peripheral connection parameter update timeout in milliseconds, range 1 to 65535 is valid. The value is a timeout used by peripheral device to wait until it starts the connection parameters update procedure to change default connection parameters. The default value is set to 5s, to comply with BT protocol specification: Core 4.2 Vol 3, Part C, 9.3.12.2 Range 1 to 65535 is valid.

CONFIG_BT_CONN_TX_MAX

Maximum number of pending TX buffers. Maximum number of pending TX buffers that have not yet been acknowledged by the controller.

CONFIG_BT_REMOTE_INFO

Enable application access to remote information. Enable application access to the remote information available in the stack. The remote information is retrieved once a connection has been established and the application is notified when this information is available through the remote_version_available connection callback.

CONFIG_BT_REMOTE_VERSION

Enable fetching of remote version. Enable this to get access to the remote version in the Controller and in the host through bt_conn_get_info(). The fields in question can be then found in the bt_conn_info struct.

CONFIG_BT_SMP_SC_ONLY

Secure Connections Only Mode. This option enables support for Secure Connection Only Mode. In this mode device shall only use Security Mode 1 Level 4 with exception for services that only require Security Mode 1 Level 1 (no security). Security Mode 1 Level 4 stands for authenticated LE Secure Connections pairing with encryption. Enabling this option disables legacy pairing.

CONFIG_BT_SMP_OOB_LEGACY_PAIR_ONLY

Force Out of Band Legacy pairing. This option disables Legacy and LE SC pairing and forces legacy OOB.

CONFIG_BT_SMP_DISABLE_LEGACY_JW_PASSKEY

Forbid usage of insecure legacy pairing methods. This option disables Just Works and Passkey legacy pairing methods to increase security.

CONFIG_BT_PRIVACY

Privacy Feature, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable local Privacy Feature support. This makes it possible to use Resolvable Private Addresses (RPAs).

CONFIG_BT_ECC

Enable ECDH key generation support. This option adds support for ECDH HCI commands.

CONFIG_BT_TINYCRYPT_ECC

Use TinyCrypt library for ECDH. If this option is used to set TinyCrypt library which is used for emulating the ECDH HCI commands and events needed by e.g. LE Secure Connections. In builds including the Bluetooth LE host, if don't set the controller crypto which is used for ECDH and if the controller doesn't support the required HCI commands the LE Secure Connections support will be disabled. In builds including the HCI Raw interface and the Bluetooth LE controller, this

option injects support for the 2 HCI commands required for LE Secure Connections so that hosts can make use of those. The option defaults to enabled for a combined build with Zephyr's own controller, since it does not have any special ECC support itself (at least not currently).

CONFIG_BT_TINYCRYPT_ECC_PRIORITY

Thread priority of ECC Task.

CONFIG_BT_HCI_ECC_STACK_SIZE

Thread stack size of ECC Task.

CONFIG_BT_RPA

Bluetooth Resolvable Private Address (RPA)

CONFIG_BT_RPA_TIMEOUT

Resolvable Private Address timeout, defaults to 900 seconds. This option defines how often resolvable private address is rotated. Value is provided in seconds and defaults to 900 seconds (15 minutes).

CONFIG_BT_SIGNING

Data signing support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables data signing which is used for transferring authenticated data in an unencrypted connection.

CONFIG_BT_SMP_APP_PAIRING_ACCEPT

Accept or reject pairing initiative. When receiving pairing request or pairing response queries, the application shall either accept proceeding with pairing or not. This is for pairing over SMP and does not affect SSP, which will continue pairing without querying the application. The application can return an error code, which is translated into an SMP return value if the pairing is not allowed.

CONFIG_BT_SMP_ALLOW_UNAUTH_OVERWRITE

Allow unauthenticated pairing for paired device. This option allows all unauthenticated pairing attempts made by the peer where an unauthenticated bond already exists. This would enable cases where an attacker could copy the peer device address to connect and start an unauthenticated pairing procedure to replace the existing bond. When this option is disabled in order to create a new bond the old bond must be explicitly deleted with bt_unpair.

CONFIG_BT_FIXED_PASSKEY

Use a fixed passkey for pairing, set passkey to fixed or not. With this option enabled, the application will be able to call the bt_passkey_set() API to set a fixed passkey. If set, the pairing_confim() callback will be called for all incoming pairings.

CONFIG_BT_BONDABLE

Bondable Mode, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for Bondable Mode. In this mode, Bonding flag in AuthReq of SMP Pairing Request/Response is set indicating the support for this mode.

CONFIG_BT_BONDING_REQUIRED

Always require bonding. When this option is enabled remote devices are required to always set the bondable flag in their pairing request. Any other kind of requests will be rejected.

CONFIG_BT_SMP_ENFORCE_MITM

Enforce MITM protection, if the macro is set to 0, feature is disabled, if 1, feature is enabled. With this option enabled, the Security Manager is set MITM option in the Authentication Requirements Flags whenever local IO Capabilities allow the generated key to be authenticated.

CONFIG_BT_OOB_DATA_FIXED

Use a fixed random number for LESC OOB pairing. With this option enabled, the application will be able to perform LESC pairing with OOB data that consists of fixed random number and confirm value. This option should only be enabled for debugging and should never be used in production.

CONFIG_BT_KEYS_OVERWRITE_OLDEST

Overwrite oldest keys with new ones if key storage is full. With this option enabled, if a pairing attempt occurs and the key storage is full, then the oldest keys in storage will be removed to free space for the new pairing keys.

CONFIG_BT_HOST_CCM

Enable host side AES-CCM module. Enables the software-based AES-CCM engine in the host. Will use the controller's AES encryption functions if available, or BT_HOST_CRYPTO otherwise.

CONFIG_BT_L2CAP_RX_MTU

Maximum supported L2CAP MTU for incoming data, if CONFIG_BT_SMP is set, range is 65 to 1300, otherwise range is 23 to 1300. Maximum size of each incoming L2CAP PDU. Range is 23 to 1300 range is 65 to 1300 for CONFIG_BT_SMP.

CONFIG_BT_L2CAP_TX_BUF_COUNT

Number of buffers available for outgoing L2CAP packets, ranging from 2 to 255. Range is 2 to 255.

CONFIG_BT_L2CAP_TX_FRAG_COUNT

Number of L2CAP TX fragment buffers, ranging from 0 to 255. Number of buffers available for fragments of TX buffers.

Warning: Setting this to 0 means that the application must ensure that queued TX buffers never need to be fragmented, that is the controller's buffer size is large enough. If this is not ensured, and there are no dedicated fragment buffers, a deadlock may occur. In most cases the default value of 2 is a safe bet. Range is 0 to 255.

CONFIG_BT_L2CAP_TX_MTU

Maximum supported L2CAP MTU for L2CAP TX buffers, if CONFIG_BT_SMP is set, the range is 65 to 2000. Otherwise, range is 23 to 2000. Range is 23 to 2000. Range is 65 to 2000 for CONFIG_BT_SMP.

CONFIG_BT_L2CAP_DYNAMIC_CHANNEL

L2CAP Dynamic Channel support. This option enables support for LE Connection oriented Channels, allowing the creation of dynamic L2CAP Channels.

CONFIG_BT_L2CAP_DYNAMIC_CHANNEL

L2CAP Dynamic Channel support. This option enables support for LE Connection oriented Channels, allowing the creation of dynamic L2CAP Channels.

Bluetooth BR/EDR support [EXPERIMENTAL] This option enables Bluetooth BR/EDR support.

CONFIG_BT_ATT_PREPARE_COUNT

Number of ATT prepares write buffers, if the macro is set to 0, feature is disabled, if greater than 1, feature is enabled. Number of buffers available for ATT prepares write, setting this to 0 disables GATT long/reliable writes.

CONFIG_BT_ATT_TX_MAX

Maximum number of queued outgoing ATT PDUs. Number of ATT PDUs that can be at a single moment queued for transmission. If the application tries to send more than this amount the calls blocks until an existing queued PDU gets sent. Range is 1 to CONFIG_BT_L2CAP_TX_BUF_COUNT.

CONFIG_BT_GATT_SERVICE_CHANGED

GATT Service Changed support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the service changed characteristic.

CONFIG_BT_GATT_DYNAMIC_DB

GATT dynamic database support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables registering/unregistering services at runtime.

CONFIG_BT_GATT_CACHING

GATT Caching support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for GATT Caching. When enabled the stack registers Client Supported Features and Database Hash characteristics which is used by clients to detect if anything has changed on the GATT database.

CONFIG_BT_GATT_CLIENT

GATT client support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the GATT Client role.

CONFIG_BT_GATT_READ_MULTIPLE

GATT Read Multiple Characteristic. Values support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the GATT Read Multiple Characteristic Values procedure.

CONFIG_BT_GAP_AUTO_UPDATE_CONN_PARAMS

Automatic Update of Connection Parameters, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option, if enabled, allows automatically sending request for connection parameters update after GAP recommended 5 seconds of connection as peripheral.

CONFIG_BT_GAP_PERIPHERAL_PREF_PARAMS

Configure peripheral preferred connection parameters. This configures peripheral preferred connection parameters. Enabling this option results in adding PPCP characteristic in GAP. If disabled it is up to application to set expected connection parameters.

CONFIG_BT_MAX_PAIRED

Maximum number of paired devices. Maximum number of paired Bluetooth devices. The minimum (and default) number is 1.

CONFIG_BT_MAX_SCO_CONN

Maximum number of simultaneous SCO connections. Maximum number of simultaneous Bluetooth synchronous connections supported. The minimum (and default) number is 1. Range 1 to 3 is valid.

CONFIG_BT_RFCOMM

Bluetooth RFCOMM protocol support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth RFCOMM support.

CONFIG_BT_RFCOMM_L2CAP_MTU

L2CAP MTU for RFCOMM frames. Maximum size of L2CAP PDU for RFCOMM frames.

CONFIG_BT_HFP_HF

Bluetooth Handsfree profile HF Role support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth HF support.

CONFIG_BT_AVDTP

Bluetooth AVDTP protocol support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth AVDTP support.

CONFIG_BT_A2DP

Bluetooth A2DP Profile [EXPERIMENTAL]. This option enables the A2DP profile.

CONFIG_BT_A2DP_SOURCE

Bluetooth A2DP profile source function. This option enables the A2DP profile Source function.

CONFIG_BT_A2DP_SINK

Bluetooth A2DP profile sink function. This option enables the A2DP profile Sink function.

CONFIG_BT_A2DP_TASK_PRIORITY

Bluetooth A2DP profile task priority. This option sets the task priority. The task is used to process the streamer data and retry command.

CONFIG_BT_A2DP_TASK_STACK_SIZE

Bluetooth A2DP profile task stack size. This option sets the task stack size.

CONFIG_BT_PAGE_TIMEOUT

Bluetooth Page Timeout. This option sets the page timeout value. Value is selected as (N * 0.625) ms.

CONFIG_BT_DIS_MODEL

Model name. The device model inside Device Information Service.

CONFIG_BT_DIS_MANUF

Manufacturer name. The device manufacturer inside Device Information Service.

CONFIG_BT_DIS_PNP

Enable PnP_ID characteristic. Enable PnP_ID characteristic in Device Information Service.

CONFIG_BT_DIS_PNP_VID_SRC

Vendor ID source, range 1 - 2. The Vendor ID Source field designates which organization assigned the value used in the Vendor ID field value. The possible values are:

- 1 Bluetooth SIG, the Vendor ID was assigned by the Bluetooth SIG
- 2 USB IF, the Vendor ID was assigned by the USB IF

CONFIG_BT_DIS_PNP_VID

Vendor ID, range 0 - 0xFFFF. The Vendor ID field is intended to uniquely identify the vendor of the device. This field is used in conjunction with Vendor ID Source field, which determines which organization assigned the Vendor ID field value. Note: The Bluetooth Special Interest Group assigns Device ID Vendor ID, and the USB Implementers Forum assigns Vendor IDs, either of which can be used for the Vendor ID field value. Device providers should procure the Vendor ID from the USB Implementers Forum or the Company Identifier from the Bluetooth SIG.

CONFIG_BT_DIS_PNP_PID

Product ID, range 0 - 0xFFFF. The Product ID field is intended to distinguish between different products made by the vendor identified with the Vendor ID field. The vendors themselves manage Product ID field values.

CONFIG_BT_DIS_PNP_VER

Product Version, range 0 - 0xFFFF. The Product Version field is a numeric expression identifying the device release number in Binary-Coded Decimal. This is a vendor-assigned value, which defines the version of the product identified by the Vendor ID and Product ID fields. This field is intended to differentiate between versions of products with identical Vendor IDs and Product IDs. The value of the field value is 0xJJMN for version JJ.M.N (JJ - major version number, M - minor version number, N - subminor version number); For example, version 2.1.3 is represented with value 0x0213 and version 2.0.0 is represented with a value of 0x0200. When upward-compatible changes are made to the device, it is recommended that the minor version number be incremented. If incompatible changes are made to the device. It is recommended that the major version number is incremented. The subminor version is incremented for bug fixes.

CONFIG_BT_DIS_SERIAL_NUMBER

Enable DIS Serial number characteristic, 1 - enable, 0 - disable. Enable Serial Number characteristic in Device Information Service.

CONFIG_BT_DIS_SERIAL_NUMBER_STR

Serial Number. Serial Number characteristic string in Device Information Service.

CONFIG_BT_DIS_FW_REV

Enable DIS Firmware Revision characteristic, 1 - enable, 0 - disable. Enable Firmware Revision characteristic in Device Information Service.

CONFIG_BT_DIS_FW_REV_STR

Firmware revision. Firmware Revision characteristic String in Device Information Service.

CONFIG_BT_DIS_HW_REV

Enable DIS Hardware Revision characteristic, 1 - enable, 0 - disable. Enable Hardware Revision characteristic in Device Information Service.

CONFIG_BT_DIS_HW_REV_STR

Hardware revision. Hardware Revision characteristic String in Device Information Service.

CONFIG_BT_DIS_SW_REV

Enable DIS Software Revision characteristic, 1 - enable, 0 - disable. Enable Software Revision characteristic in Device Information Service.

CONFIG_BT_DIS_SW_REV_STR

Software revision Software revision characteristic String in Device Information Service.

CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE

System work queue stack size.

CONFIG_SYSTEM_WORKQUEUE_PRIORITY

System work queue priority.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_TYPE

HCI transport interface type.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_INSTANCE

HCI transport interface instance number.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_SPEED

HCI transport interface rate. Configures the interface speed, for example, the default interface is h4, the speed to 115200

CONFIG_BT_HCI_TRANSPORT_TX_THREAD

Whether enable HCI transport TX thread.

CONFIG_BT_HCI_TRANSPORT_RX_THREAD

Whether enable HCI transport RX thread.

CONFIG_BT_HCI_TRANSPORT_RX_STACK_SIZE

HCI transport RX thread stack size.

CONFIG_BT_HCI_TRANSPORT_TX_STACK_SIZE

HCI transport TX thread stack size.

CONFIG_BT_HCI_TRANSPORT_TX_PRIO

HCI transport TX thread priority.

CONFIG_BT_HCI_TRANSPORT_RX_PRIO

HCI transport RX thread priority.

CONFIG_BT_MSG_QUEUE_COUNT

Message number in message queue.

Rework Guide for EdgeFast Bluetooth Protocol Abstraction Layer

Hardware Rework Guide for MIMXRT1170-EVKB and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

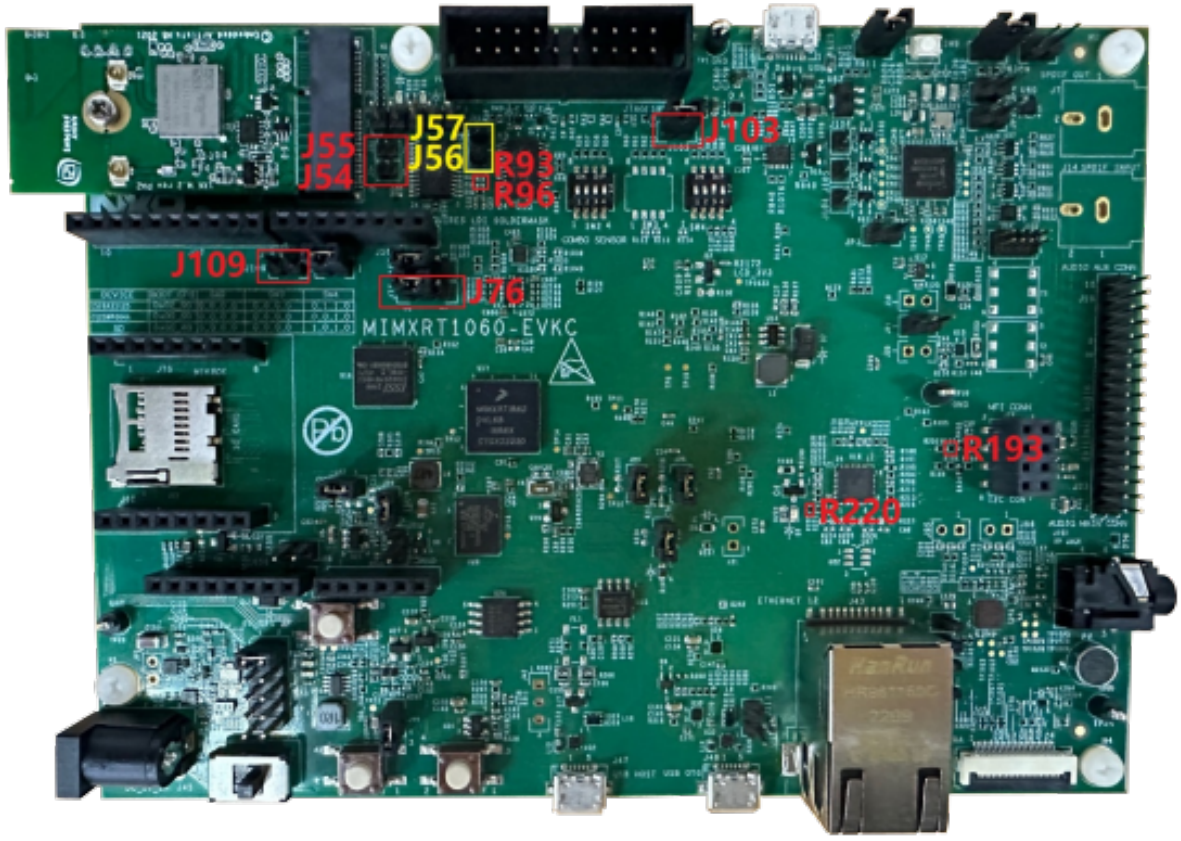
- **HCI UART rework**

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

- **PCM interface rework**

1. Remove J54 and J55, connect J56 and J57.
2. Remove R220.
3. Connect J103.

Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1170-EVKB and Murata 2EL M.2 Module Hardware Rework Guide for MIMXRT1170-EVKB and Murata

2EL M.2 Module

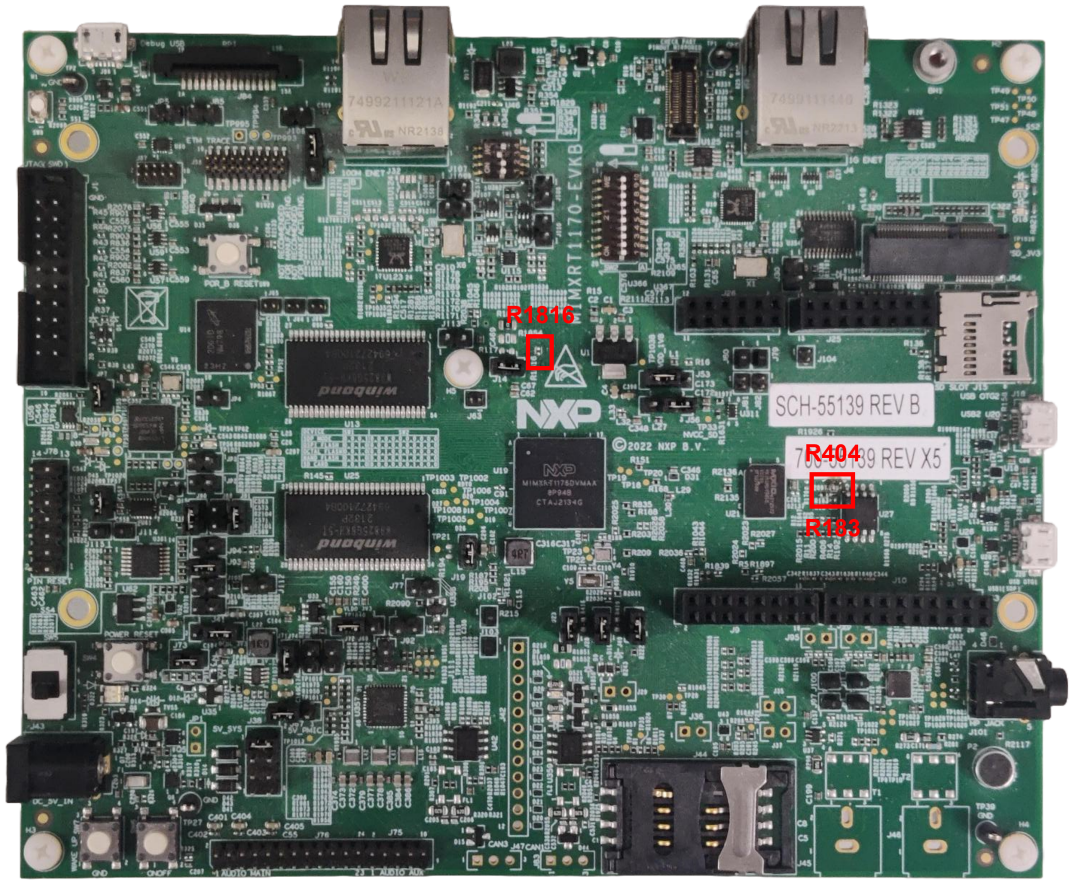
This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

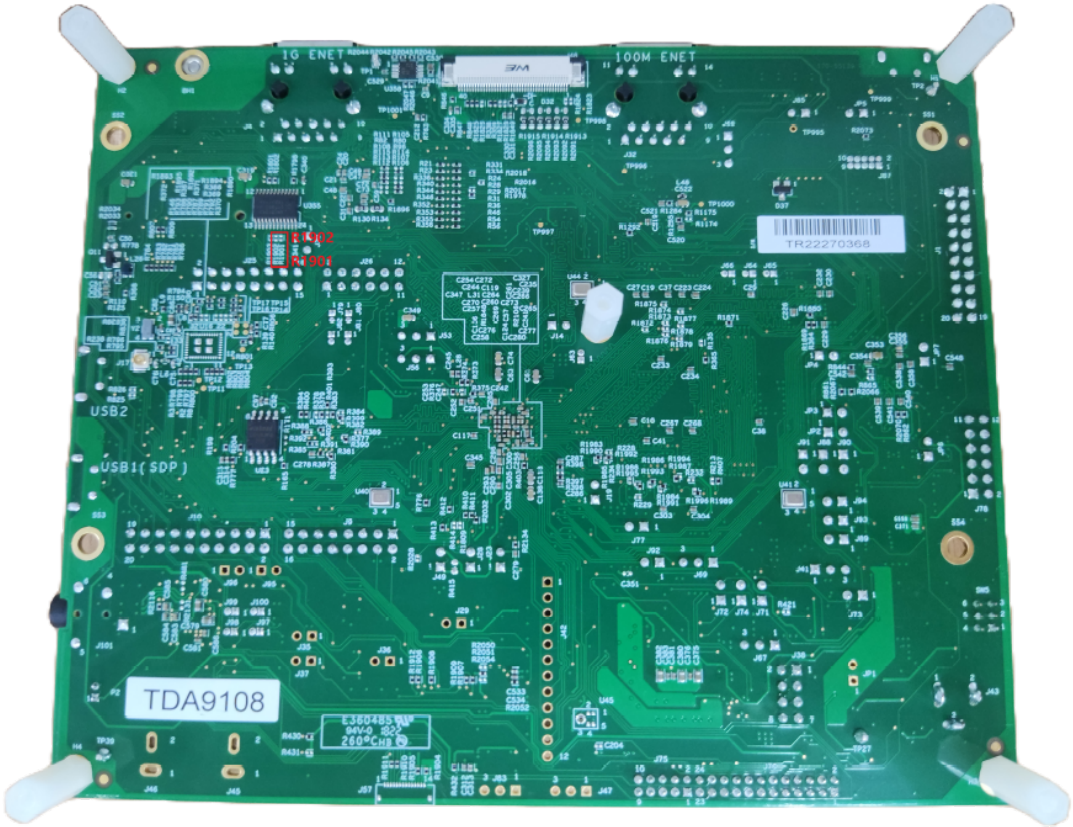
The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

Hardware rework

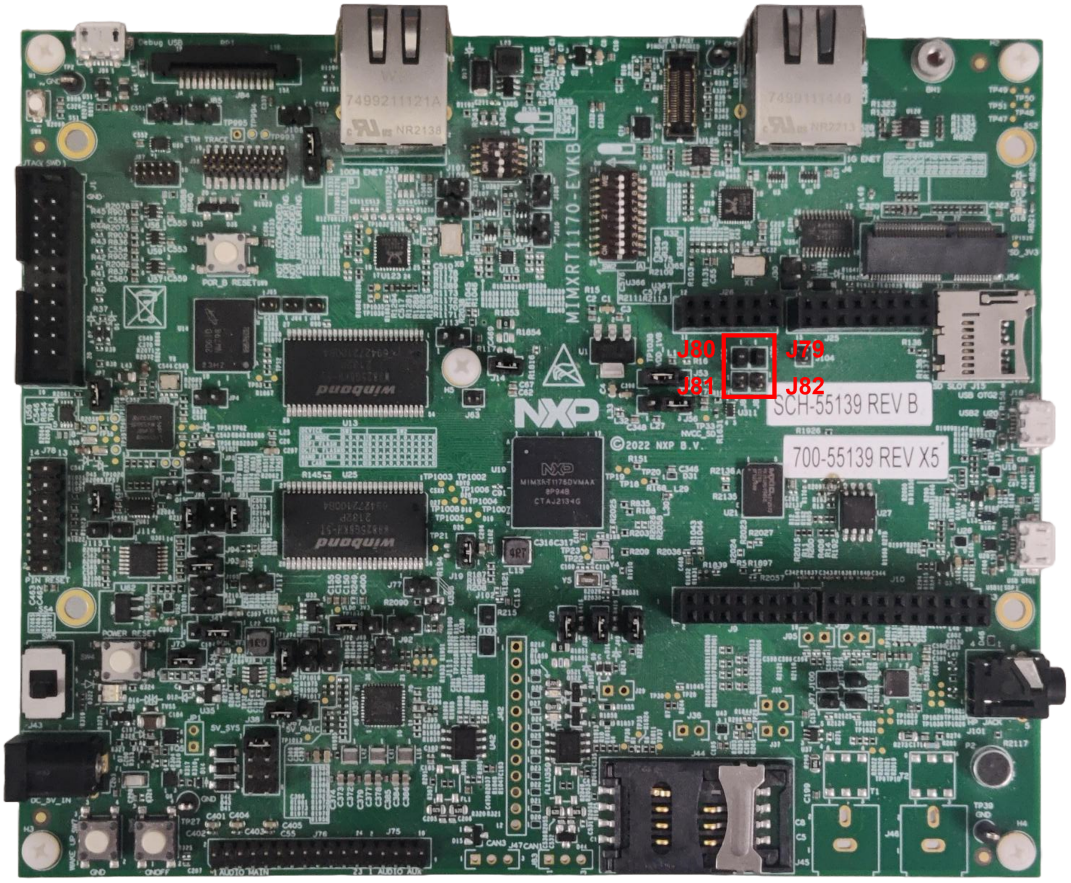
- **HCI UART rework**
  1. Remove resistors R183 and R1816.
  2. Solder 0 ohm resistor to R404, R1901, and R1902.

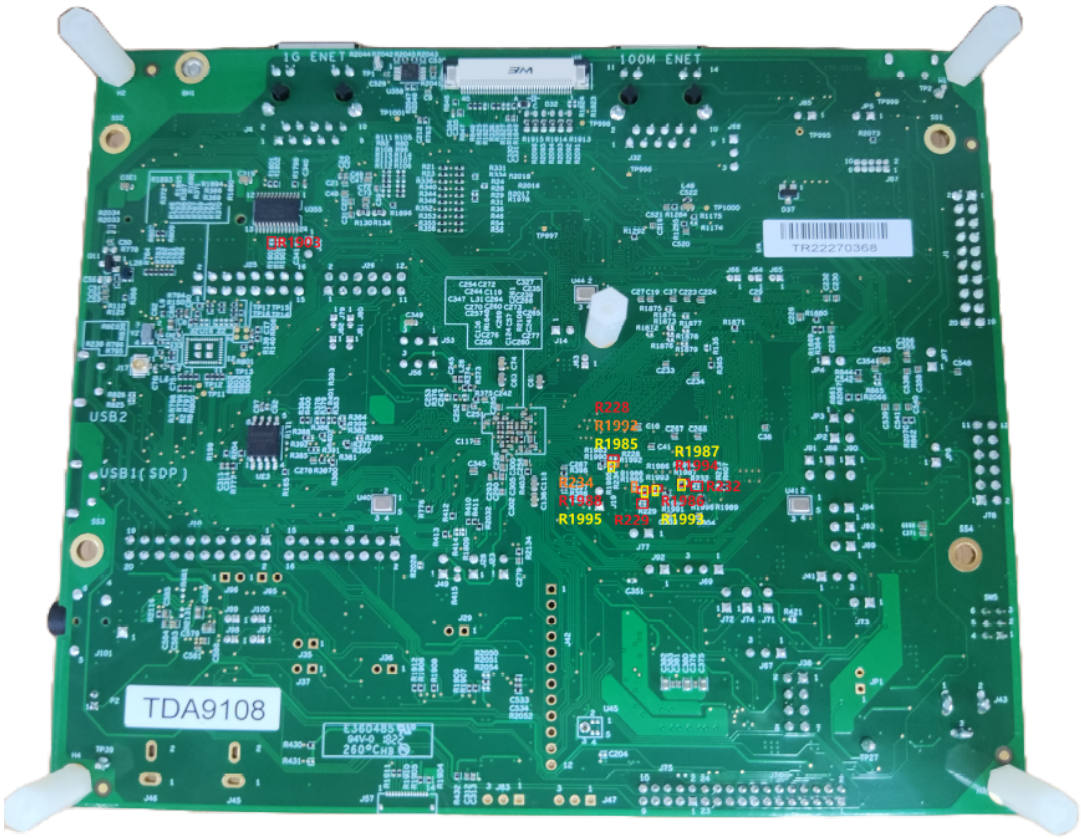




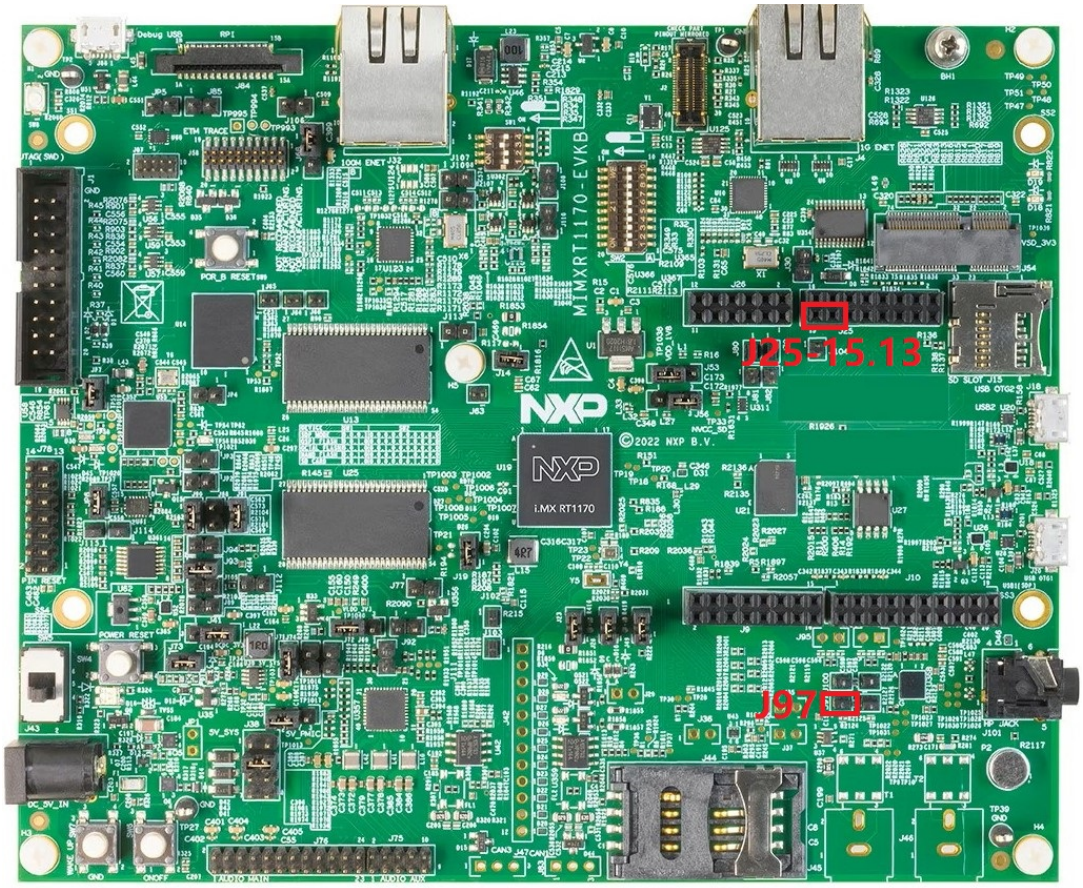
- **PCM interface rework**
  1. Disconnect header J79 and J80.

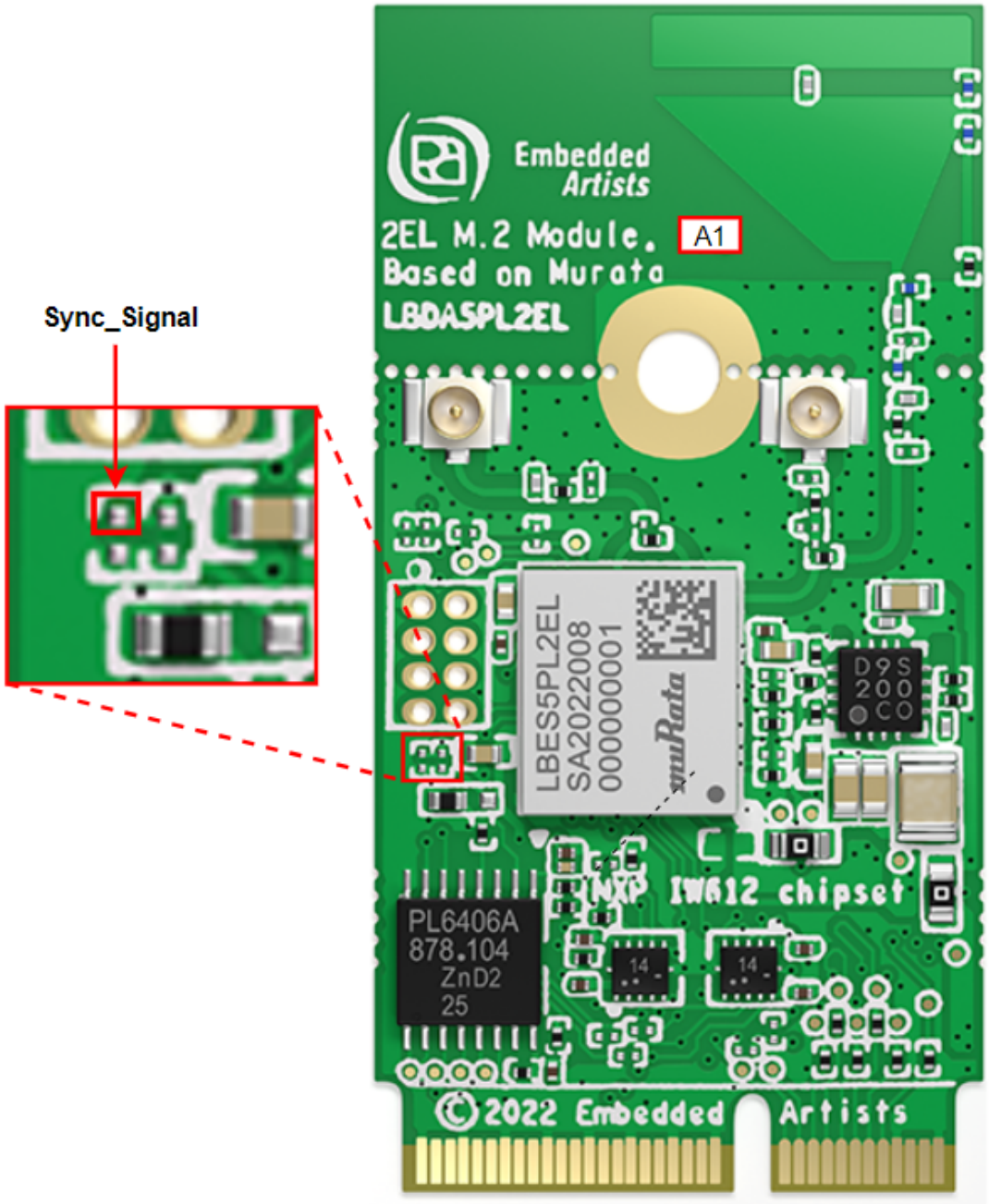
2. Connect header J81 and J82.
3. Remove resistors R1985, R1986, R1987, R1988, R1992, R1993, R1994, and R1995.
4. Solder 0 ohm resistor to R228, R229, R232, R234, and R1903.





- LE Audio Synchronization interface rework (only used on sink side)
  1. Connect J25-15 with J97.
  2. Connect J25-13 with 2EL's GPIO_27


MIMXRT1170-EVKB
i.MX RT1170
J25 15.13
J97



Parent topic:*Hardware Rework Guide for MIMXRT1170-EVKB and Murata 2EL M.2 Adapter*

Hardware Rework Guide for MIMXRT685-EVK and AW-AM457-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-AM457-uSD. The AW-AM457-uSD user guide is available here.The hardware rework has one part:

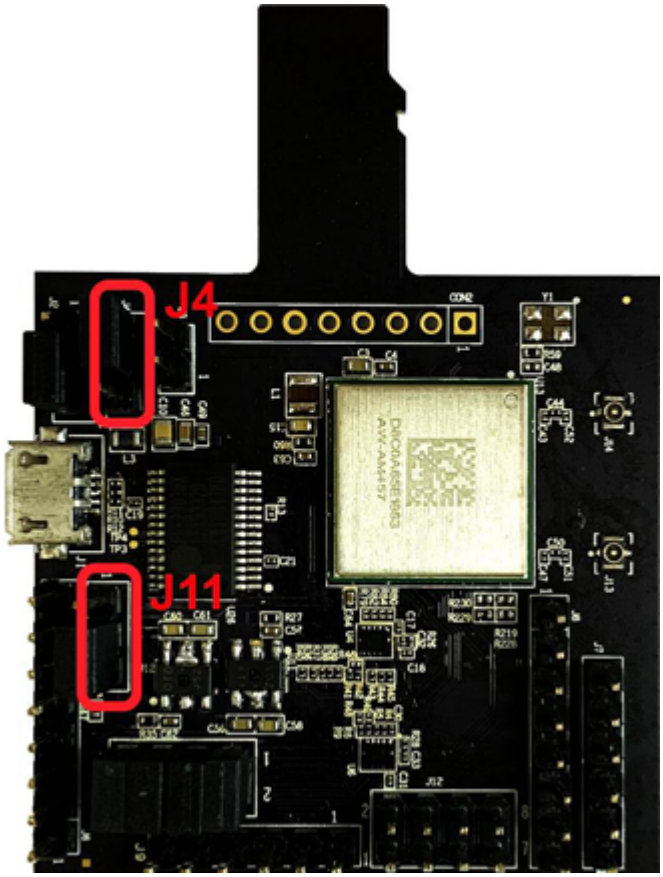
- HCI UART rework

Hardware rework **HCI UART rework**

- R398 move from 1-2 to 2-3
- JP12 2-3
- Connect the pins of two boards as the following table.

Pin Name	AW-AM457-uSD	i.MX RT685	PIN NAME	GPIONAME of i.MX RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	US-ART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND





Jumper Settings:

- Connect J4[2-3] for VIO 3.3 V supply
- Connect J11[2-3] for VIO_SD 3.3 V supply

PCM interface rework

Connect the pins of two boards as the following table.

Pin Name	AW-AM457-uSD	i.MX RT685	PIN NAME of I.MX RT685	GPIONAME of I.MX RT685
PCM_IN	J9 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J9 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J9 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J9 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J9 (pin 6)	J29 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-AM457-uSD*

Hardware Rework Guide for MIMXRT685-EVK and AW-CM358-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-CM358-uSD. The AW-CM358-uSD user guide is available here. The hardware rework has one part:

- HCI UART rework

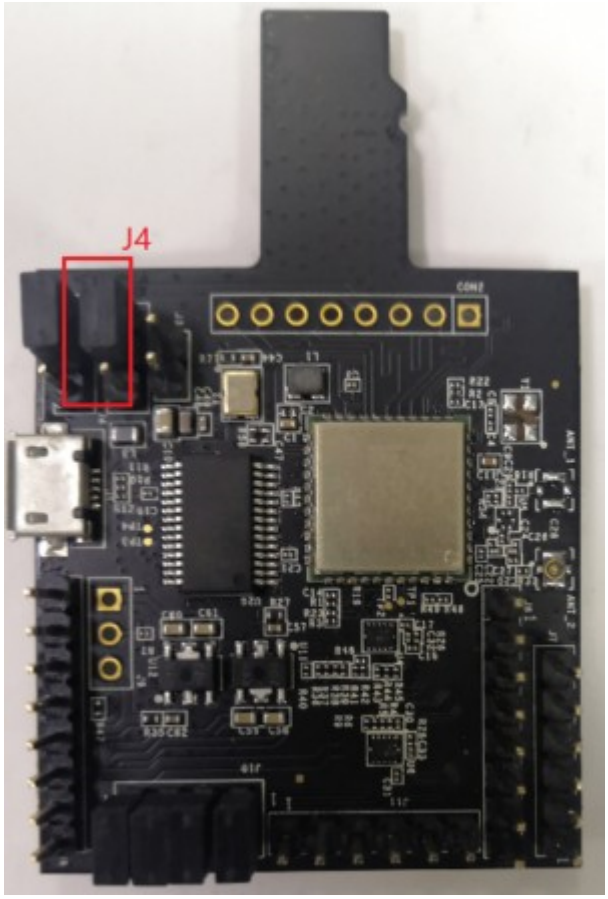
Hardware rework **HCI UART rework**

R398 move from 1-2 to 2-3.

Connect the pins of two boards as the following table.

Pin Name	AW-CM358-USD	i.MXRT685	PIN NAME	GPIONAME of RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND





Jumper Setting:

Connect J4[1-2] for VIO 1.8 V supply.

PCM interface rework

Connect the pins of two boards as the following table.

Pin Name	AW-CM358-USD	i.MX RT685	PIN NAME of RT685	GPIONAME of RT685
PCM_IN	J11 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J11 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J11 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J11 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J11 (pin 5)	J29 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-CM358-uSD*

Hardware Rework Guide for MIMXRT685-EVK and AW-AM510-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-AM510-uSD. The AW-AM510-uSD user guide is available here. The hardware rework has one part:

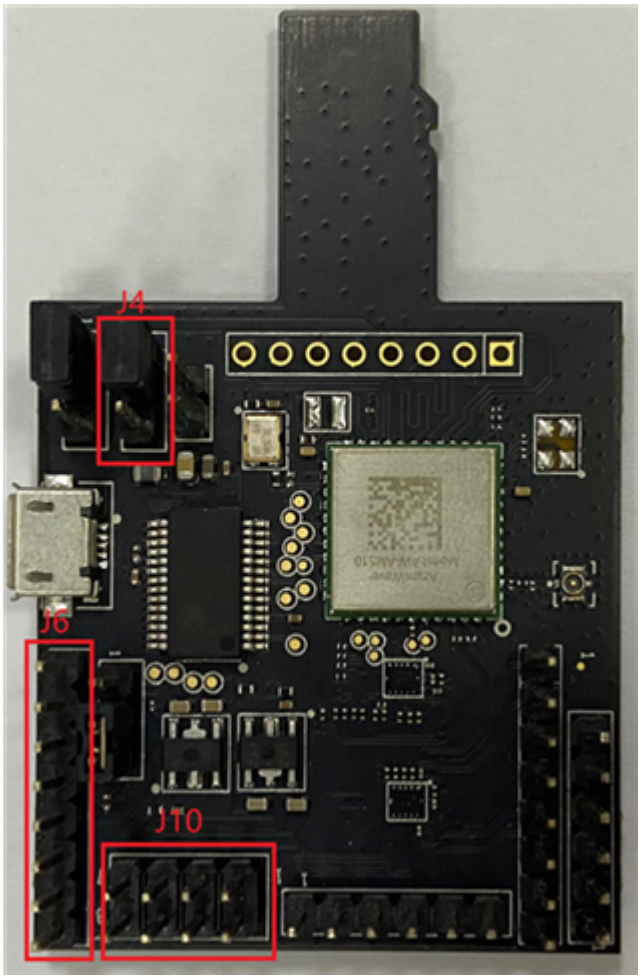
- HCI UART rework

Hardware rework

- **HCI UART rework**

Connect the pins of two boards as the following table.

Pin Name	AW-AM510-uSD	i.MXRT685	PIN NAME	GPIO NAME of RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND



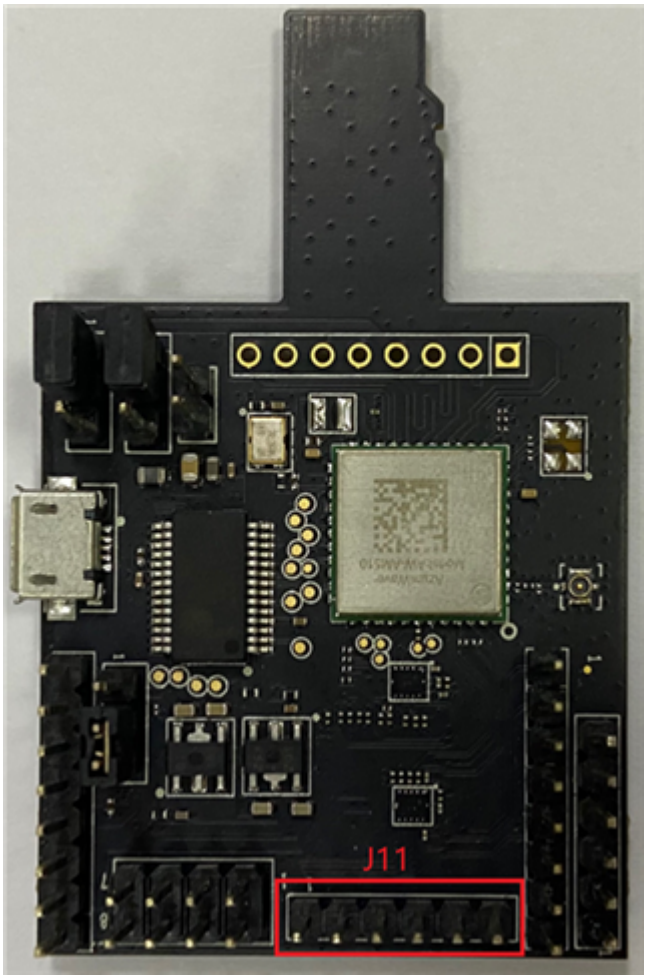
Jumper Setting:

– Connect J4[2-3] for VIO 3.3 V supply

- **PCM interface rework**

Connect the pins of two boards as the following table.

PIN NAME	AW-AM510-USD	i.MX RT685	PIN NAME of RT685	GPIONAME of RT685
PCM_IN	J11 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J11 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J11 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J11 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J11 (pin 6)	J29 (pin 7)	GND	GND



Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-AM510-uSD*

Hardware Rework Guide for MIMXRT685-EVK and Murata uSD-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and the Murata uSD-M.2 adapter. For details on the Murata uSD-M.2 Adapter, see Murata's uSD-M.2 webpage.

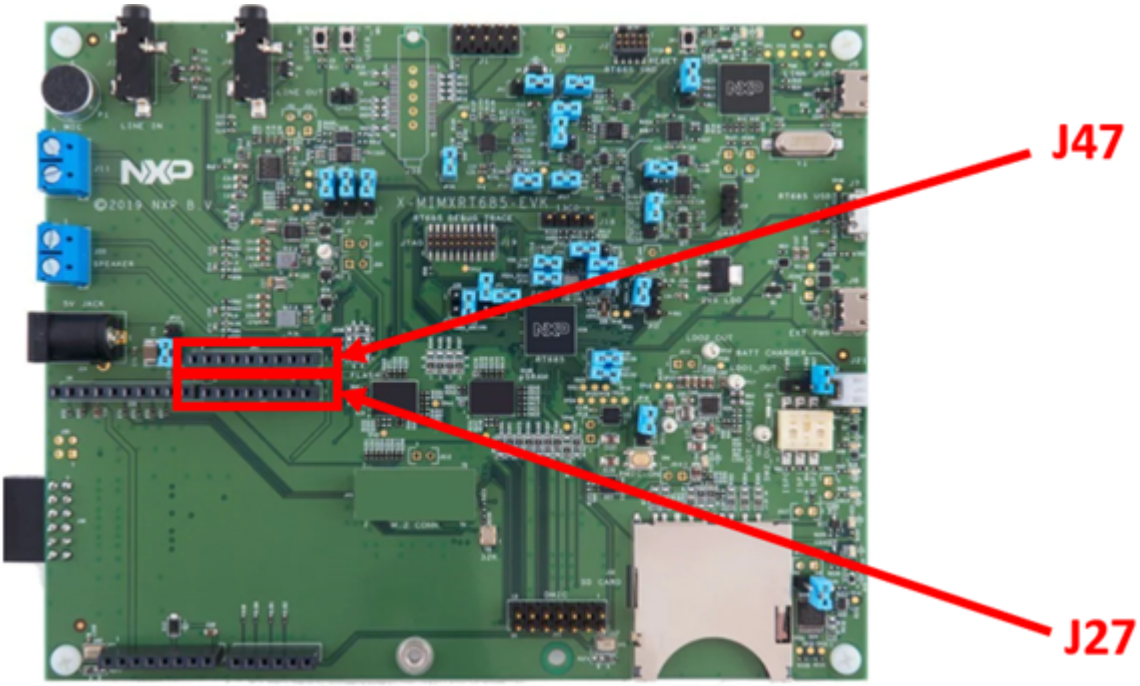
The hardware rework has one part:

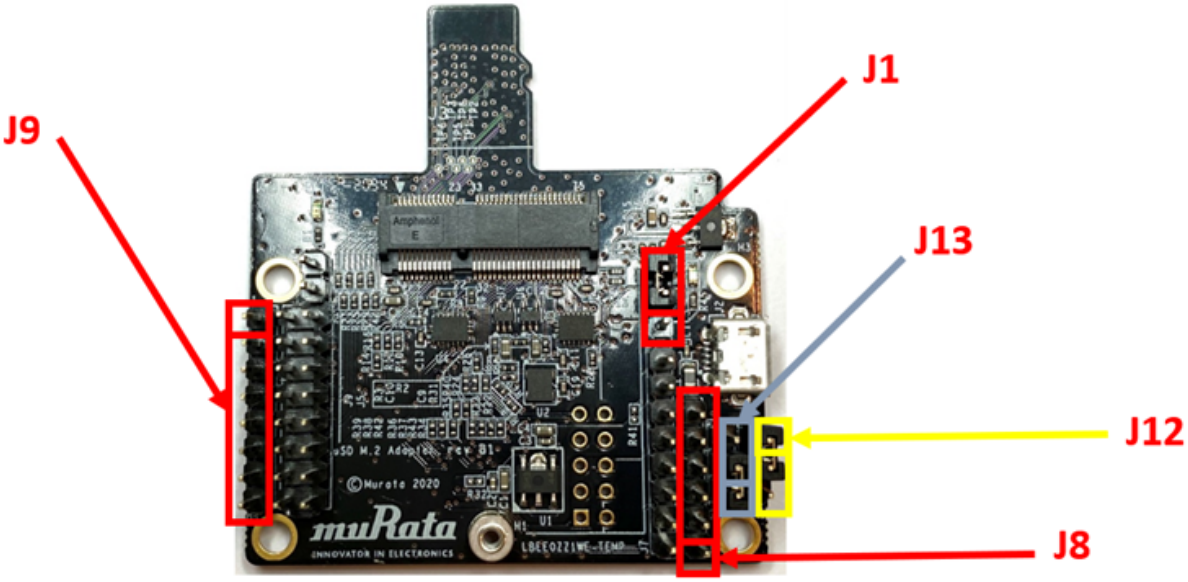
- HCI UART rework

Hardware rework **HCI UART rework :**

- JP12 2-3
- Connect the pins of two boards as the following table using jumper cables included in Murata's uSD-M.2 Adapter kit.

Pin name	uSD-M.2 adapter pin	i.MX RT685 pin	Pin name of RT685	GPIO name of RT685
BT_UART_TXD_HO	J9 (pin 1)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
BT_UART_RXD_HO	J9 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
BT_UART_RTS_HO	J8 (pin 3)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
BT_UART_CTS_HO	J8 (pin 4)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1





Murata uSD-M.2 jumper settings:

- Both J12 and J13 = 1-2 (WLAN-SDIO = 1.8 V; and BT-UART and WLAN/BT-CTRL = 3.3 V)
- J1 = 2-3 (3.3 V from uSD connector)

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and Murata uSD-M.2 Adapter*

Hardware Rework Guide for MIMXRT685-AUD-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the Edgefast Bluetooth PAL on the NXP i.MX MIMXRT685-AUD-EVK board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has one part:

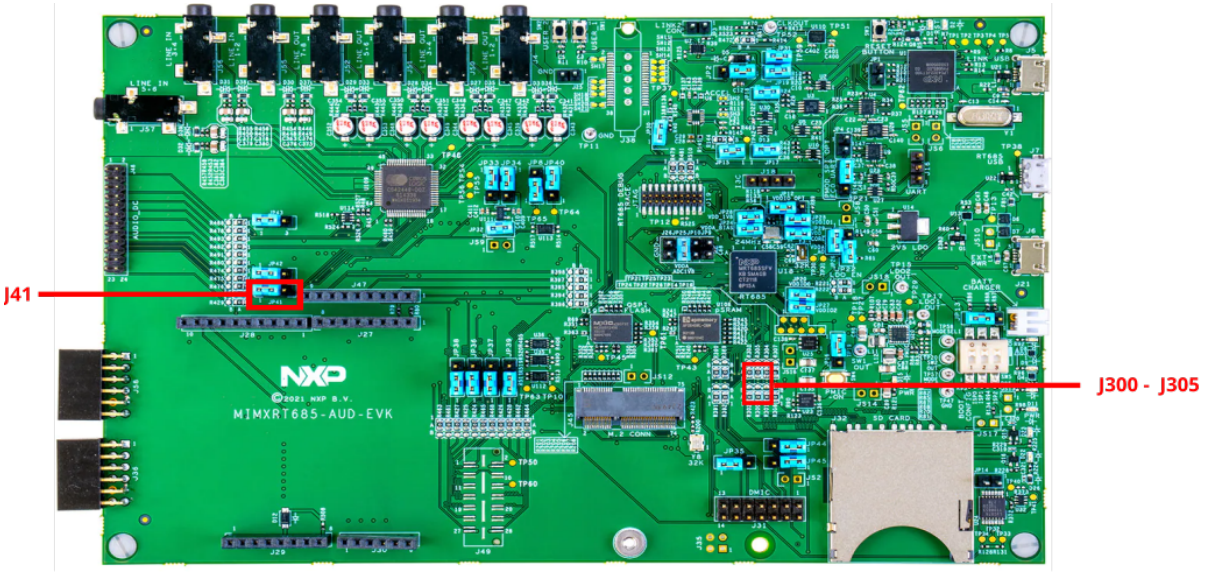
- HCI UART rework

Hardware rework HCI UART rework:

Mount R300~R305 A-B

Jumper Setting:

- Connect JP41[2-3]



Parent topic:*Hardware Rework Guide for MIMXRT685-AUD-EVK and Murata M.2 Module*

Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 2EL M.2 Module

This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL for low power feature on the NXP i.MX MIMXRT595-EVK board and the Murata's 2EL - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

The hardware rework has three parts:

- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework

- **Debug console serial rework**

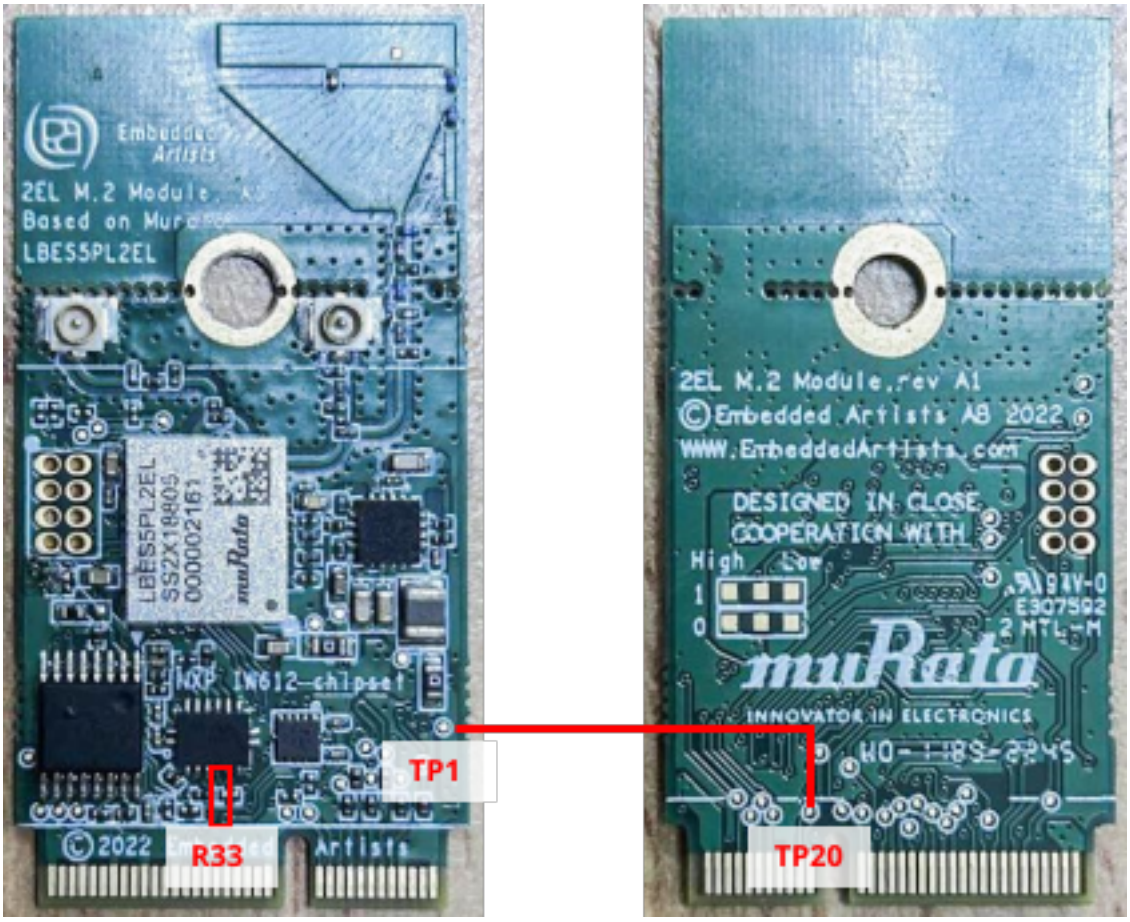
  For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

- **Host wake-up controller pin rework**:

  For details, refer Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module.

- **Controller wake-up host pin rework:**

1. Remove resistors R709 on MIMXRT595-EVK,
2. Solder 0K ohm resistor on R33 of Murata 2EL M.2 Module
3. Solder 10K ohm resistor on the Murata 2EL M.2 Module between TP1 and TP20.



Parent topic:*Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 2EL M.2 Module*

Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT595-EVK board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

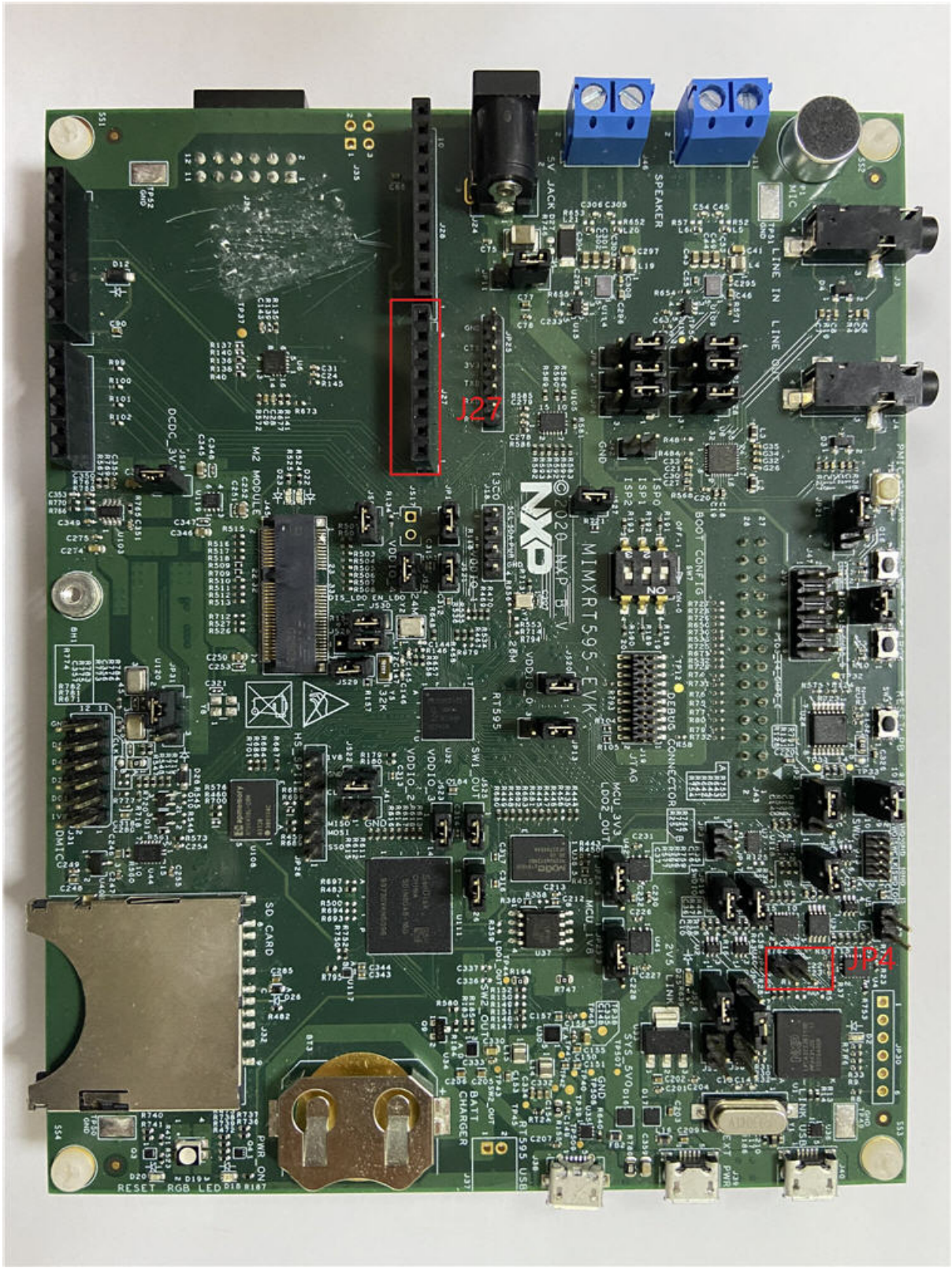
The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

No special rework is required, except the following to enable the debug port.

- JP4 1-2.
- J27 1 - TX of USB to serial converter
- J27 2 - RX of USB to serial converter



Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*

Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL for low power feature on the NXP i.MX MIMXRT595-EVK board and the Murata's 1XK - direct M.2 connection to Embedded Artists EAR00385 (1XK) M.2 modules.

The hardware rework has three parts:

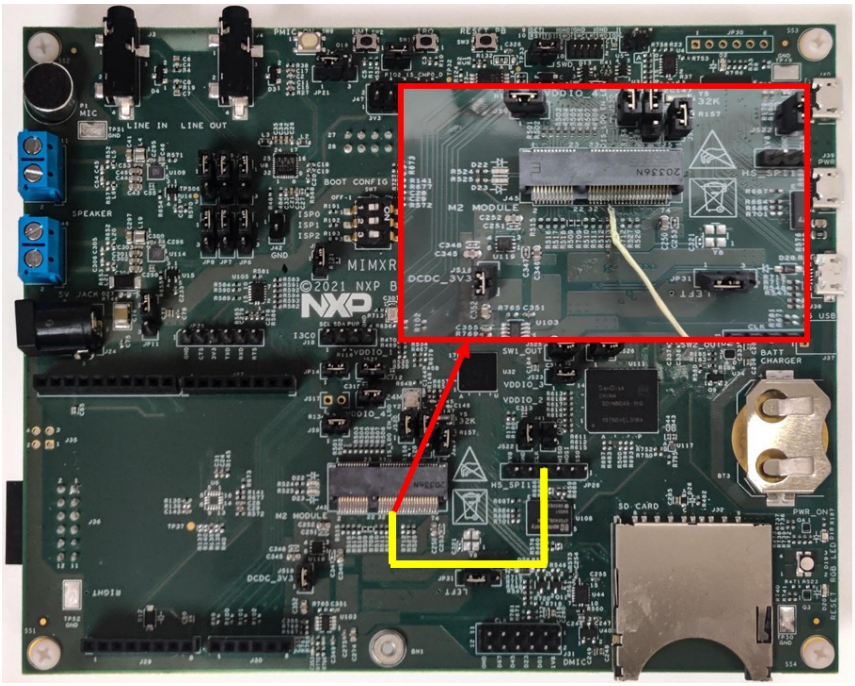
- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework **Debug console serial rework:**

For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

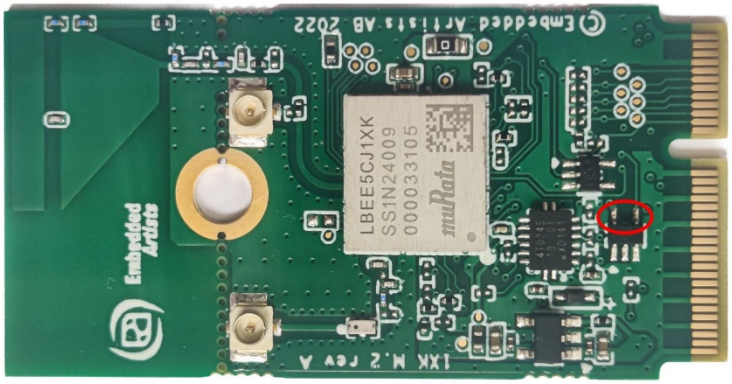
Host wake-up controller pin rework:

Connect M.2 (pin 42) to JP26 (pin 4) with a wire.



Controller wake-up host pin rework:

1. Remove resistors R709 on MIMXRT595-EVK.
2. Solder 10K ohm resistor on the Murata 1XK M.2 Module at the location shown in the following figure.



Parent topic:*Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module*

Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT595-EVK board and AW-AM510MA. The AW-AM510MA user guide is available here. The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

No special rework is required, except the following to enable the debug port.

- Connect J39 with external power.
- Connect JP4 1-2.
- J27 1 — TX of USB to serial converter.
- J27 2 — RX of USB to serial converter.



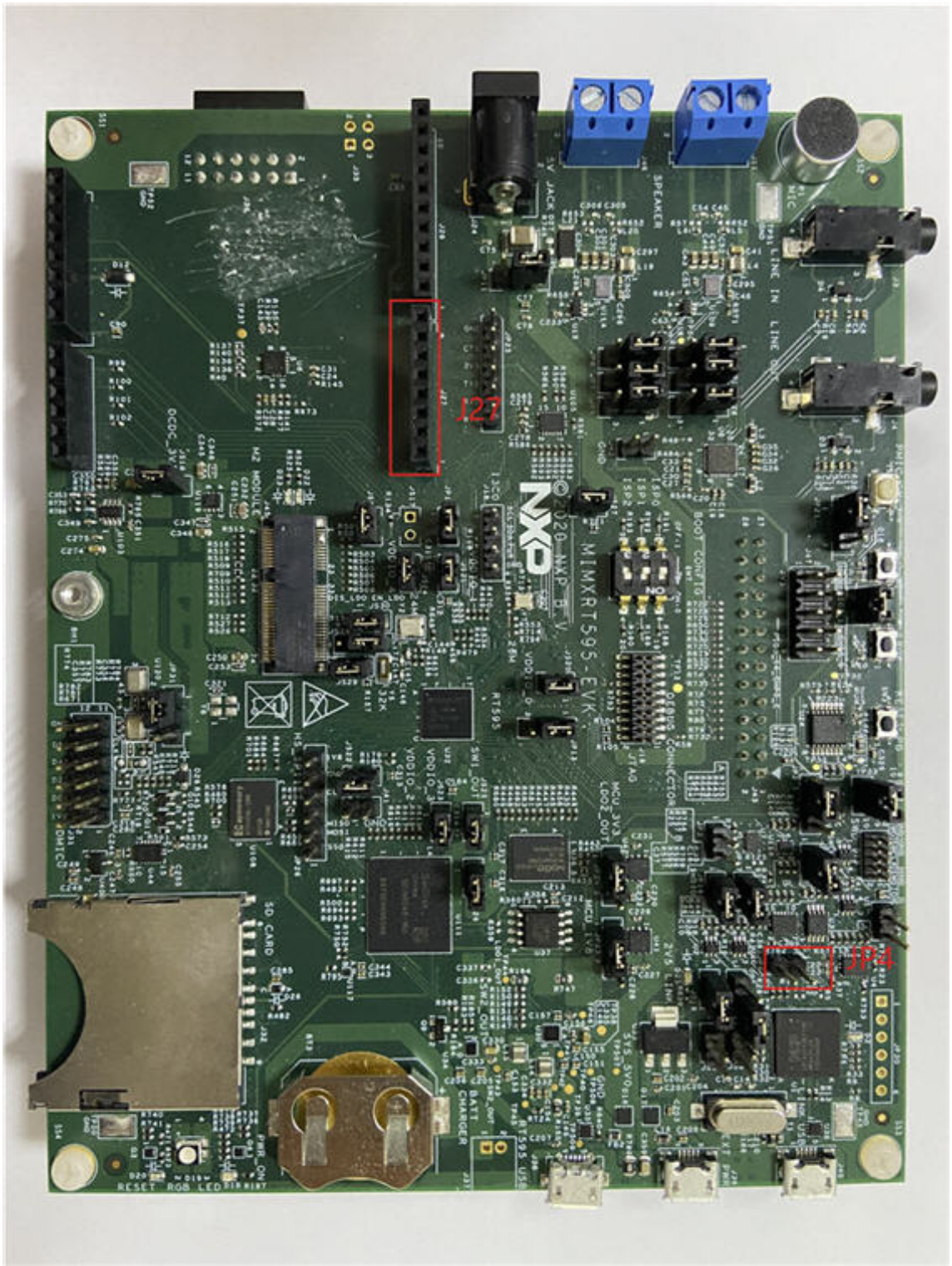
Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA*

Hardware Rework Guide for MIMXRT595-EVK and AW-CM358MA This section is a brief hardware rework guidance of the Ethermind Bluetooth stack on the NXP i.MX MIMXRT595-EVK board and AW-CM358MA. The AW-CM358MA user guide is available here. The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

- Connect J39 with external power.
- JP4 1-2
- J27 1 - TX of USB to serial converter
- J27 2 - RX of USB to serial converter



Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and AW-CM358MA*

Hardware Rework Guide for MIMXRT1040-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the Edgefast Bluetooth PAL on the NXP i.MX MIMXRT1040-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework
- Wake pin rework

Hardware rework

1. HCI UART rework
    - Solder R93 and R96
2. PCM interface rework
    - Solder R70 and R79; remove R76 and R86; Connect J80.
3. Wake pin rework
    - When using 2LL M.2 module, remove R456 and R457 to avoid the module has an impact on boot configuration.

Note: Make sure to disconnect J80 when debugging. Otherwise, the debugger downloading fails.

Parent topic:*Hardware Rework Guide for MIMXRT1040-EVK and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1060-EVKC and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

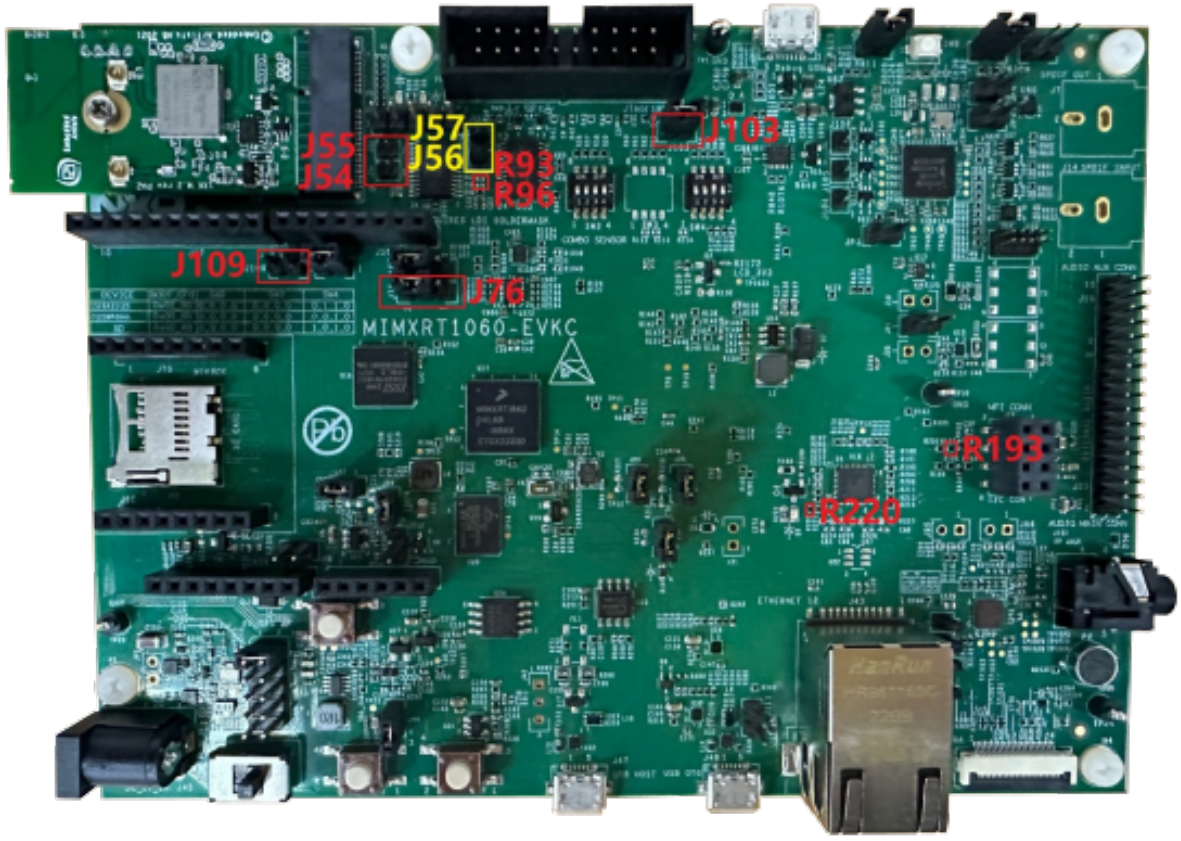
- **HCI UART rework**

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

- **PCM interface rework**

1. Remove J54 and J55, connect J56 and J57.
2. Remove R220.
3. Connect J103.

Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1060-EVKC and Murata 2EL M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1060-EVKC and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

Hardware rework

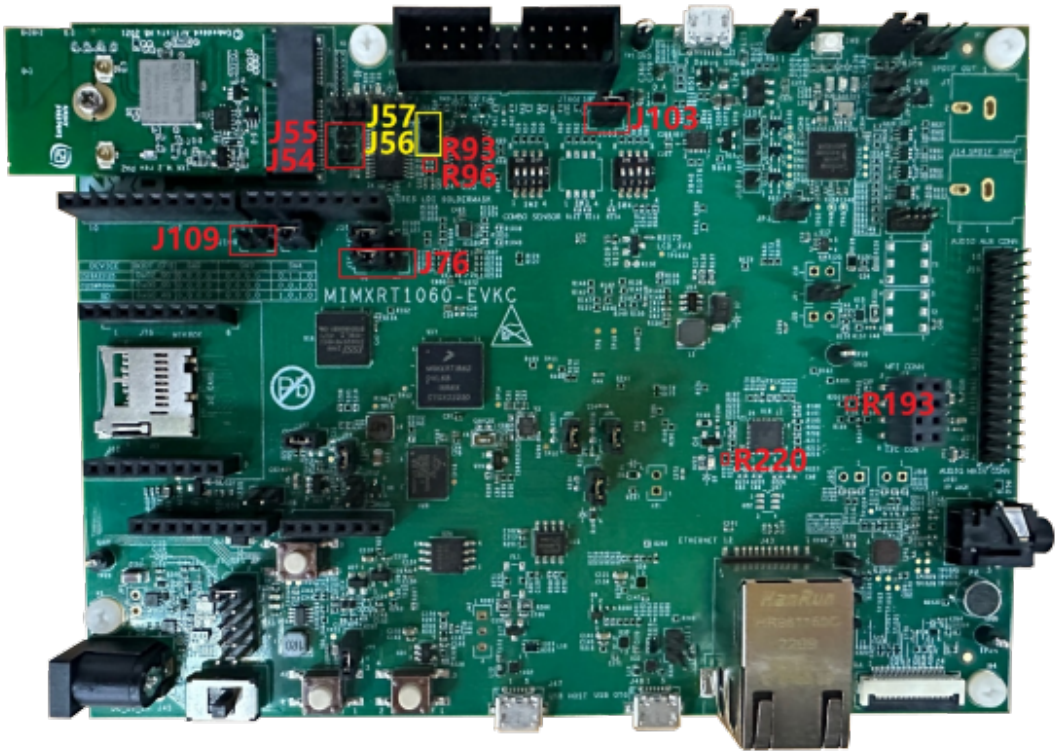
- HCI UART rework

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

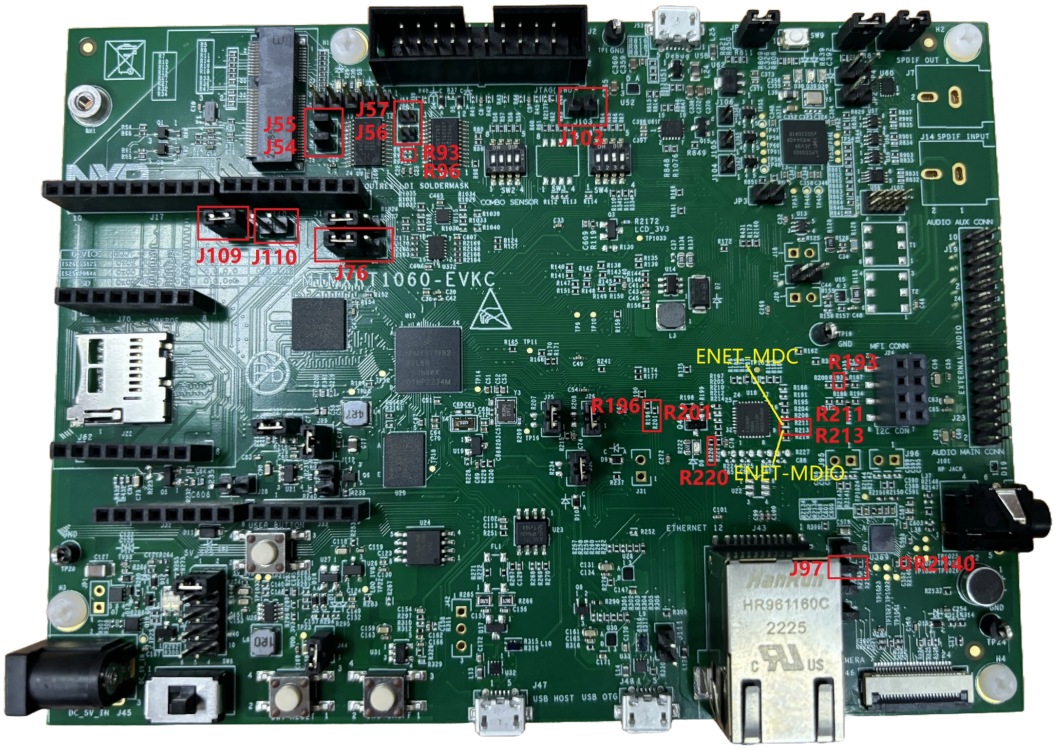
- PCM interface rework

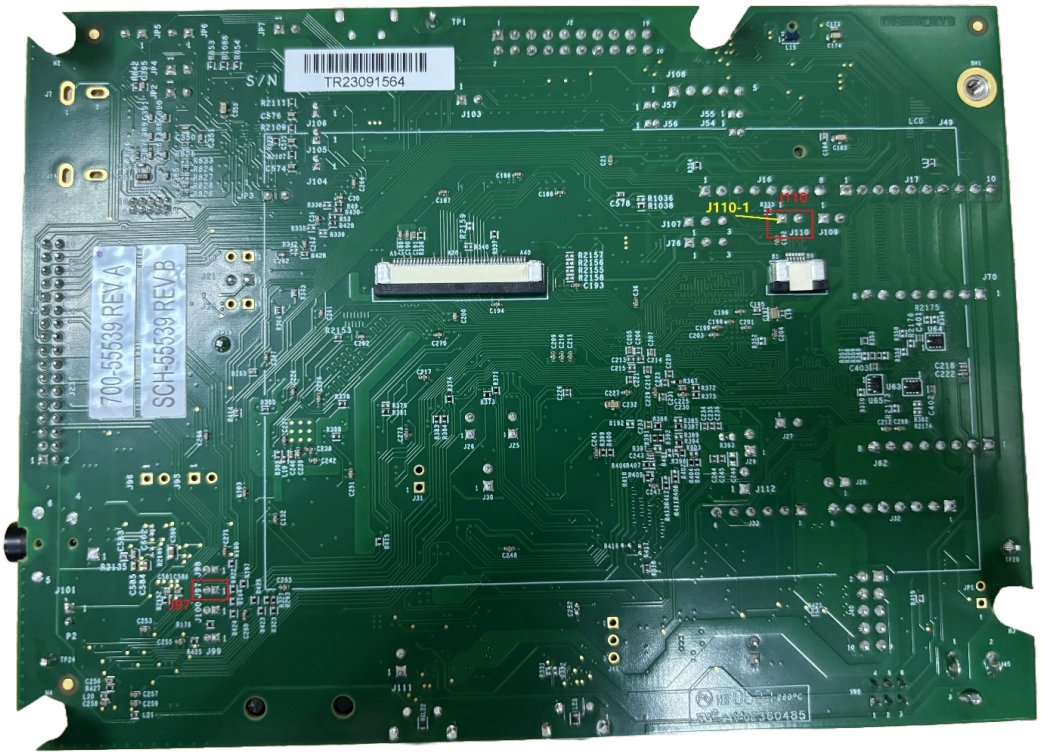
1. Remove J54 and J55, connect J56, and J57.
2. Remove R220.
3. Connect J103.

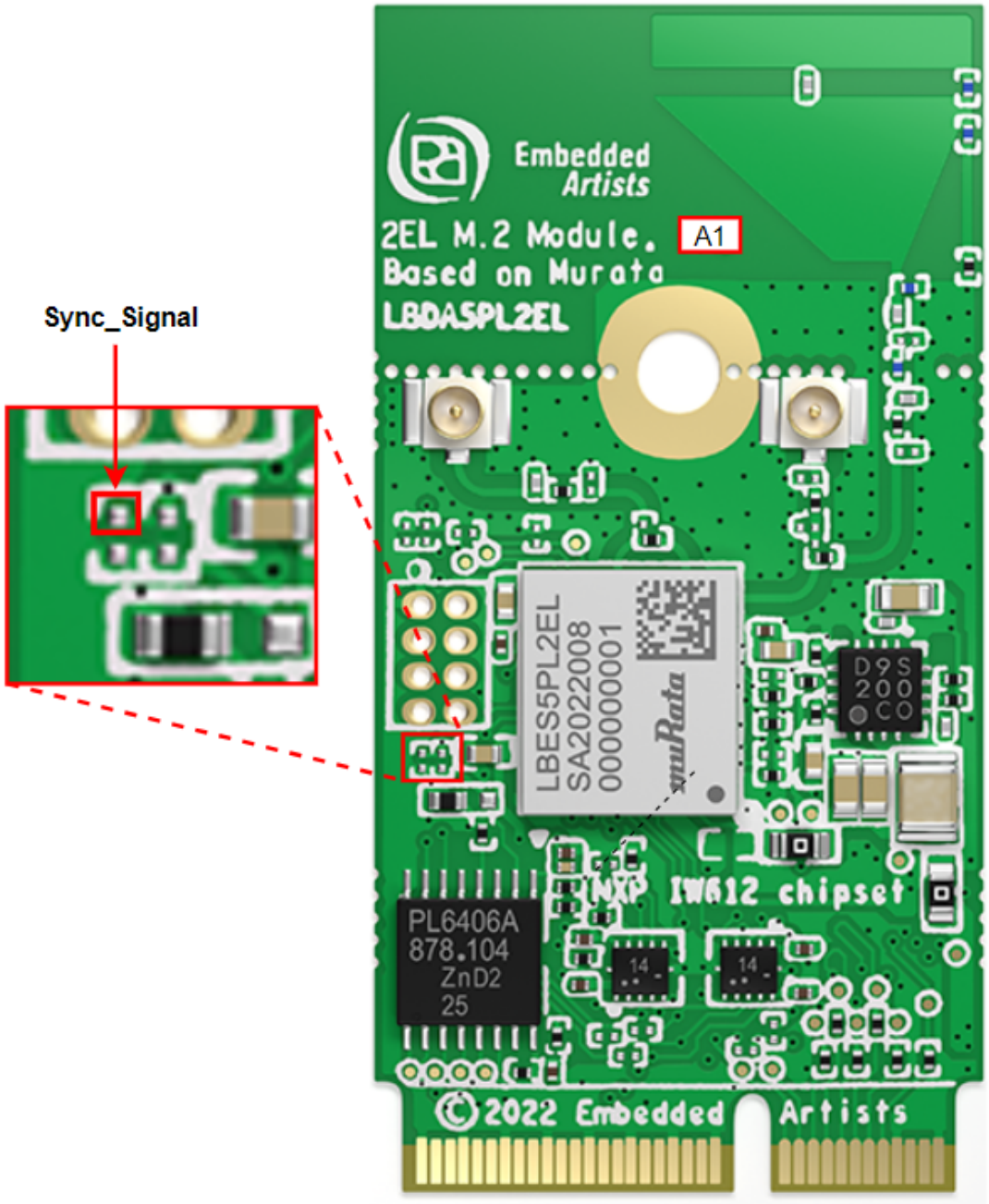
   Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



- LE Audio Synchronization interface rework (only used on sink side)
  1. Remove J110 jumper cap.
  2. Remove R196, R201, R213, and R211.
  3. Connect J110-1 (GPT2_CLK) to R2140 (SAI_MCLK).
  4. Connect ENET_MDIO (GPT2_CAP1) with J97 (SAI_SW).
  5. Connect ENET_MDC (GPT2_CAP2) with 2EL's GPIO_27 (Sync Signal).




S/N
TR23091564
700-55539 REV.A
SCH-55539 REV.B
J110-1
J110
J107



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata 2EL M.2 Adapter*

Hardware Rework Guide for MCXN547-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MCXN547-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

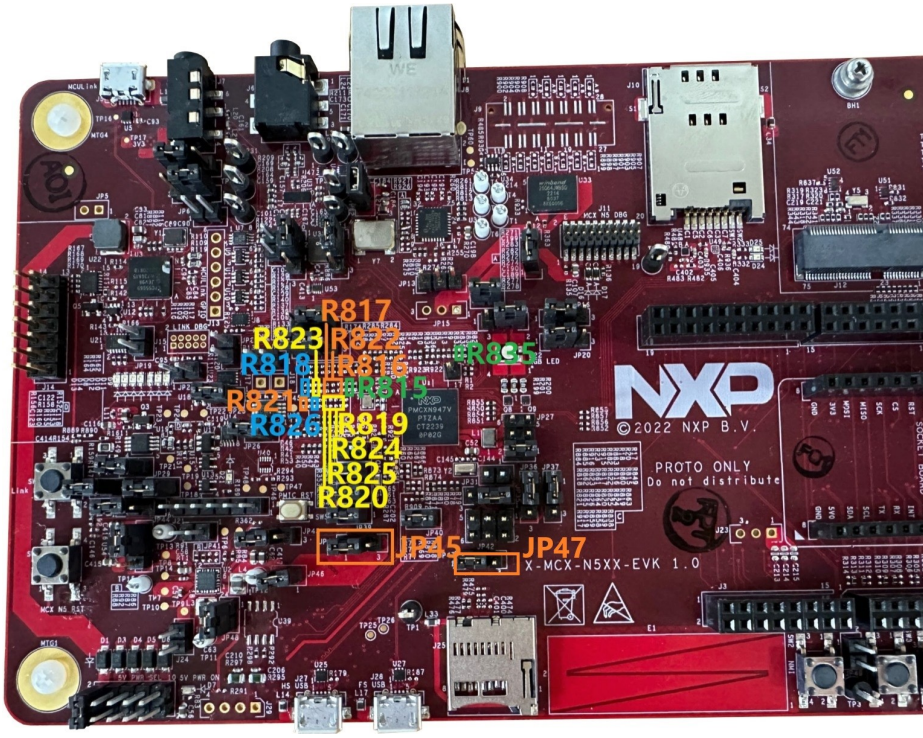
The hardware rework consists of two parts:

- M.2 UART interface
- M.2 SDIO interface

Hardware rework

- M.2 UART interface rework

  - Mount R835
  - Connect JP45 2-3 to supply 1.8V for GPIO4
- M.2 SDIO interface rework
  - Connect JP47 2-3 to supply 1.8V for GPIO2
  - Remove R818, connect R823
  - Remove R819, connect R824
  - Remove R817, connect R822
  - Remove R815, connect R816
  - Remove R820, connect R825
  - Remove R821, connect R826



Hardware Rework Guide for MCXN947-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MCXN947-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

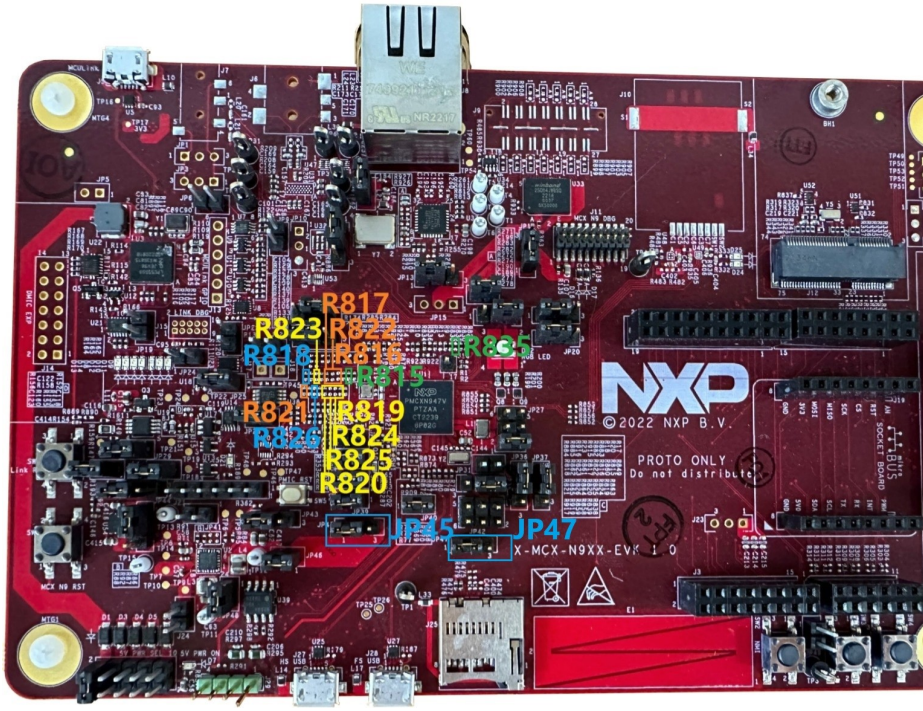
The hardware rework consists of two parts:

- M.2 UART interface
- M.2 SDIO interface

Hardware rework

- M.2 UART interface rework
  - Mount R835
  - Connect JP45 2-3 to supply 1.8V for GPIO4
- M.2 SDIO interface rework
  - Connect JP47 2-3 to supply 1.8V for GPIO2

  - Remove R818, connect R823
  - Remove R819, connect R824
  - Remove R817, connect R822
  - Remove R815, connect R816
  - Remove R820, connect R825



  - Remove R821, connect R826

Hardware Rework Guide for IMXRT1050-EVKB and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP IMXRT1050-EVKB board and the Murata 1XK,1ZM and 2EL solution - direct M.2 connection to Embedded Artists' EAR00385 (1XK) , EAR00364 (1ZM) or EAR00409 (2EL)M.2 modules. The hardware rework consists of three parts:

- Murata uSDM
- HCI UART rework

Hardware rework

- Murata uSD-M.2 jumper settings
  - J12 = 1-2: WLAN-SDIO & BT-PCM = 1.8 V
  - J13 = 1-2: BT-UART & WLAN/BT-CTRL = 3.3 V
  - J1 = 2-3: 3.3 V from uSD connector
- HCI UART interface rework

  Connect the TX/RX/RTS/CTS pins of the two boards as show in Table 1 using the jumper cables included in the Murata's uSD-M.2 Adapter kit as shown in the following table.

Pin name	uSD-M.2 adapter pin	i.MX RT1050-EVKB pin	Pin name of RT1050-EVKB	GPIO name of RT1050-EVKB
BT_UART_TXD_	J9 (pin 1)	J22 (pin 1)	LPUART3_RXD	GPIO_AD_B1_07
BT_UART_RXD_	J9 (pin 2)	J22 (pin 2)	LPUART3_TXD	GPIO_AD_B1_06
BT_UART_RTS_]	J8 (pin 3)	J23 (pin 3)	LPUART3_CTS	GPIO_AD_B1_04
BT_UART_CTS_]	J8 (pin 4)	J23 (pin 4)	LPUART3_RTS	GPIO_AD_B1_05
GND	J7 (pin 7)	J25 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for IMXRT1050-EVKB and Murata M.2 Module*

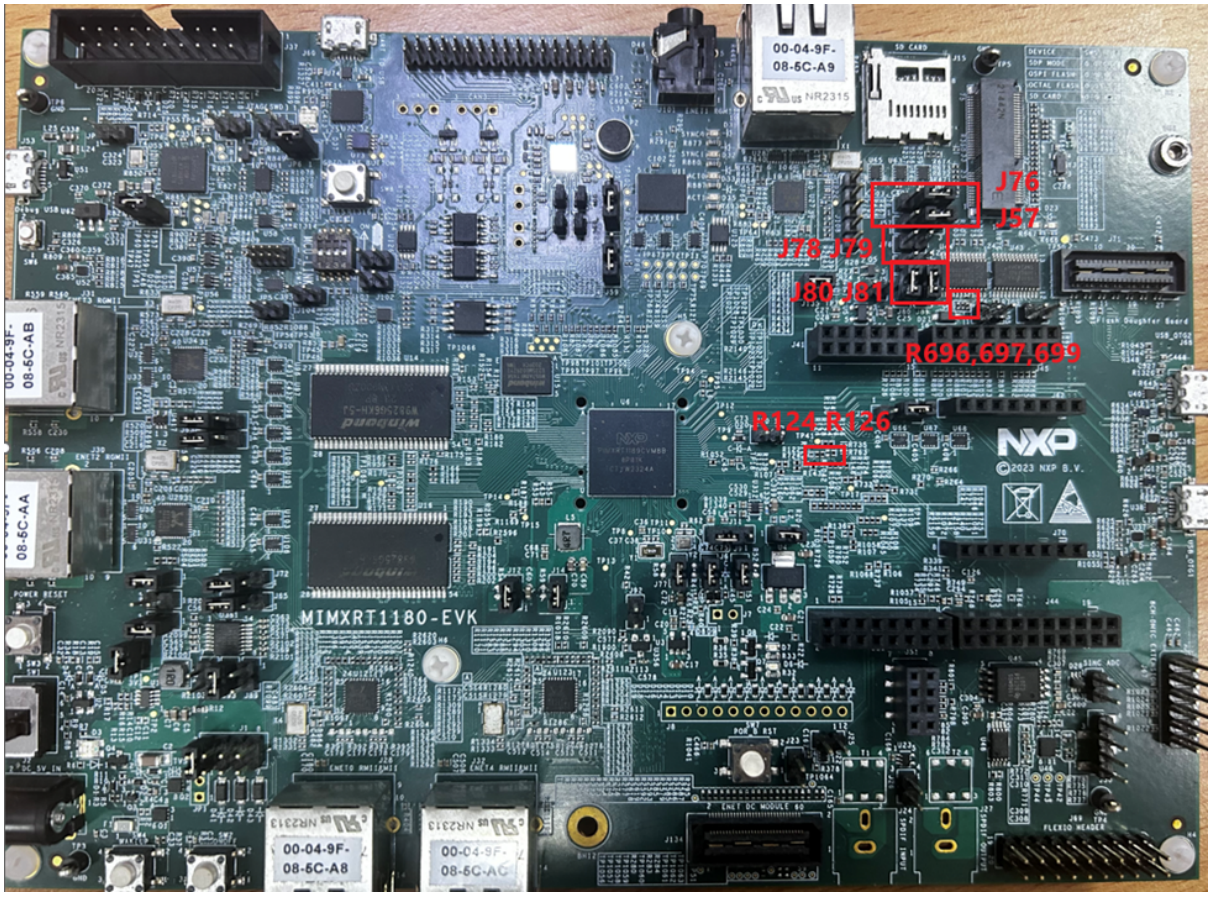
Hardware Rework Guide for MIMXRT1180 and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MIMXRT1180 board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework consists of two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

- HCI UART rework:
  - Remove: R124,R126
  - Mount R696, R697
  - Connect J57 [2-3], J76 [2-3]
- PCM interface rework
  - Mount R699
  - Disconnect J78 J79
  - Connect J80 J81



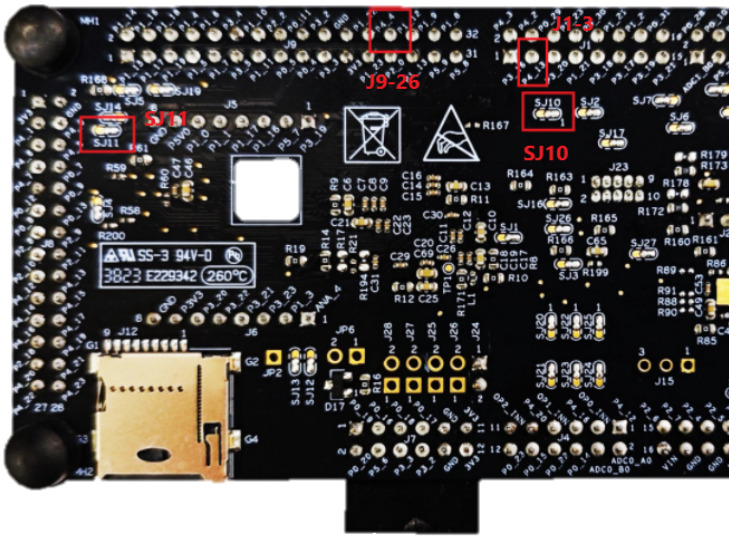
Hardware Rework Guide for FRDM-MCXN947 and X-FRDM-WIFI-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and X-FRDM-WIFI-M.2 or the Murata's 2LL EAR00500 (2LL) M.2 modules solution.

The hardware rework consists of one part:

- UART interface rework

Hardware rework

- UART interface rework
  - Remove SJ11 1-2, connect SJ11 2-3
  - Remove SJ10 1-2, connect J1-3 to J9-26
- X-FRDM-WIFI-M.2 jumper setting
  - Connect J8(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J24(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J19(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J25(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J15(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J16(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J17(On X-FRDM-WIFI-M.2) for 1.8V



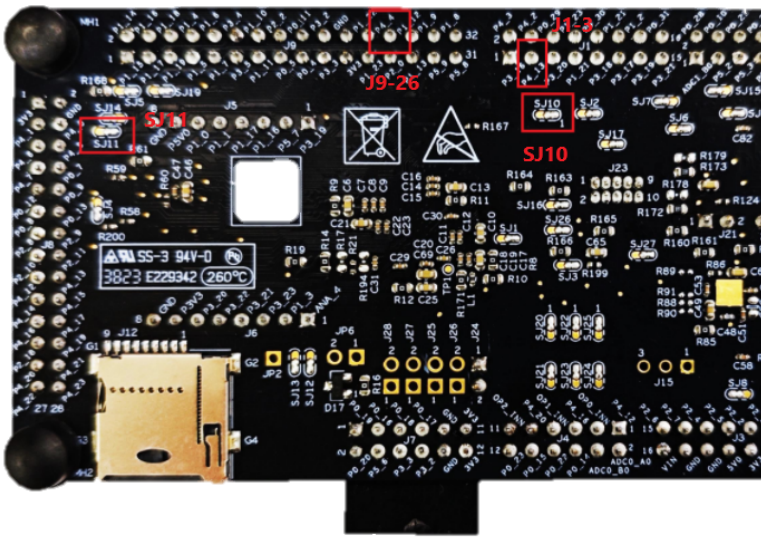
– Connect J18(On X-FRDM-WIFI-M.2) for 3.3V

Hardware Rework Guide for FRDM-MCXN947 and FRDM-IW416-AW-AM510 This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and FRDM-IW416-AW-AM510 board. The hardware rework consists of two parts:

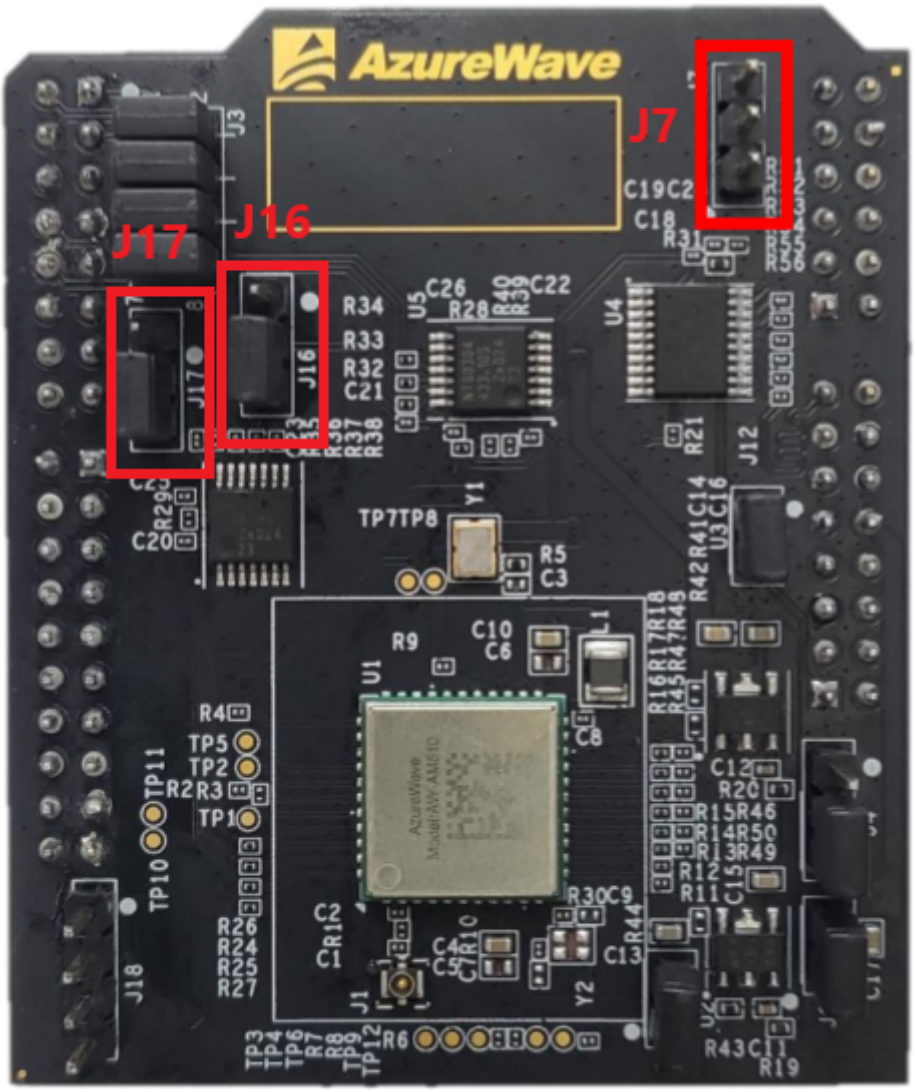
- UART interface rework
- FRDM-IW416-AW-AM510

Hardware rework

- UART interface rework
  - Remove SJ11 1-2, connect SJ11 2-3
  - Remove SJ10 1-2, connect J1-3 to J9-26
- FRDM-IW416-AW-AM510 jumper setting
  - Connect J16 2-3 for 3.3V supply
  - Connect J17 2-3 for 3.3V UART voltage level



– Connect J7 2-3 for 3.3V SDIO voltage level



Enabling Additional EdgeFast Bluetooth Protocol Abstraction Layer Examples on RT1064

Introduction NXP supports Bluetooth/Bluetooth Low Energy on RT1060EVK and RT1060EVKC. RT1064 has the same MCU die with RT1060EVK and RT1060EVKC and therefore it is possible to migrate the examples.

This document takes *peripheral_ht* as an example and describes the steps to migrate EdgeFast examples from RT1060EVK to RT1064 (based on SDK 2.13.0) and from RT1060EVKC to RT1064 (based on SDK 2.14.0) with different toolchains including IAR, Arm GCC, and MDK.

Migrate examples from RT1060EVK to RT1064 This topic describes the Common steps and the steps to migrate with the IAR, Arm GCC, and MDKtoolchains.

Common steps

1. Download *SDK_2.13.0_EVK-MIMXRT1060* and *SDK_2.13.0_EVK-MIMXRT1064*.
2. Copy the following folders from RT1060EVK package to RT1064 package: <install_dir>/components/internal_flash/ <install_dir>/middleware/edgefast_bluetooth/ <install_dir>/middleware/wireless/.

3. Create a folder named edgefast_bluetooth_examples/ under <rt1064_install_dir>/boards/evkmimxrt1064/.
4. Copy the entire folder from <rt1060evk_install_dir>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/ to < rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/.
5. Copy *clock_config.[c/h]* and *board.c* from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/ to <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/ to replace the previous files.
6. Add #define EDGEFAST_BT_LITTLEFS_MFLASH 1 in <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht /app_config.c.
7. Make the following changes in <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth/peripheral_ht/board.h.

```
73 #define BOARD_FLASH_SIZE (0x800000U)
```

```
73 #define BOARD_FLASH_SIZE (0x400000U)
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes.

File name	Previous item	New item
peripheral_ht.ewp	1060 1062	1064 1064

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.

```
47 define symbol m_interrupts_start       = 0x60002000;
48 define symbol m_interrupts_end         = 0x600023FF;
49
50 define symbol m_text_start             = 0x60002400;
51 define symbol _ROM_END_                = 0x6057FFFF;
66 define exported symbol m_boot_hdr_conf_start = 0x60000000;
67 define symbol m_boot_hdr_ivt_start           = 0x60001000;
68 define symbol m_boot_hdr_boot_data_start     = 0x60001020;
69 define symbol m_boot_hdr_dcd_data_start      = 0x60001030;
95 BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash secto
96 BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x60800000 - EDGEFAST_BT_
97 ***/
```

```
47 define symbol m_interrupts_start       = 0x70002000;
48 define symbol m_interrupts_end         = 0x700023FF;
49
50 define symbol m_text_start             = 0x70002400;
51 define symbol _ROM_END_                = 0x7017FFFF;
66 define exported symbol m_boot_hdr_conf_start = 0x70000000;
67 define symbol m_boot_hdr_ivt_start           = 0x70001000;
68 define symbol m_boot_hdr_boot_data_start     = 0x70001020;
69 define symbol m_boot_hdr_dcd_data_start      = 0x70001030;
95 BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash sect
96 BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x70400000 - EDGEFAST_BT
97 ***/
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.
2. Rename the following files.

Path	Previous name	**New name**
<rt1064_install_dir>/middleware/wireless/ethermind/	middleware_edgefast_bluetooth_cmake	middleware_edgefast_bluetooth_k32w061_contro cmake

3. Make following changes.

File name	Previous item	New item
config.cmake	1060	1064
	1062	1064
flags.cmake	1062	1064
CMakeLists.txt	1060	1064
	1062	1064

4. *mflash* is used in RT1064 instead of flash_adapter,therefore, comment include(component_flexspi_nor_flash_adapter_rt1064_MIMXRT1064) in *CMakeLists.txt*.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.

```
_ROM_START_ = 0x60002400;
_ROM_END_ = 0x6057FFFF;
EDGEFAST_BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x60800000 - EDG
/*** littleFS configuration End ***/


HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x10
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x04
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400

/* Specify the memory areas */
MEMORY
{
  m_flash_config        (RX)  : ORIGIN = 0x60000000, LENGTH
  m_ivt                 (RX)  : ORIGIN = 0x60001000, LENGTH
  m_interrupts          (RX)  : ORIGIN = 0x60002000, LENGTH
```

```
_ROM_START_ = 0x70002400;
_ROM_END_ = 0x7017FFFF;
EDGEFAST_BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x70400000 - EDG
/*** littleFS configuration End ***/


HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x10
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x04
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400

/* Specify the memory areas */
MEMORY
{
  m_flash_config        (RX)  : ORIGIN = 0x70000000, LENGTH
  m_ivt                 (RX)  : ORIGIN = 0x70001000, LENGTH
  m_interrupts          (RX)  : ORIGIN = 0x70002000, LENGTH
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

MDK

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

2. Make following changes.

File name	Previous item	New item
peripheral_ht.uvprojx	1060	1064
	1062	1064

3. Copy evkmimxrt1064_flexspi_nor.ini from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/mdk/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

4. Rename MIMXRT1062xxxxx_flexspi_nor as MIMXRT1064xxxxx_flexspi_nor and make the following changes.

```
#define m_flash_config_start            0x60000000
#define m_flash_config_size             0x00001000

#define m_ivt_start                     0x60001000
#define m_ivt_size                      0x00001000

#define m_interrupts_start              0x60002000
#define m_interrupts_size               0x00000400

#define m_text_start                    0x60002400
#define _ROM_END_                       0x6057FFFF
 EDGEFAST_BT_LITTLEFS_STORAGE_START_ADDRESS  (_ROM_END_ + 1)
 EDGEFAST_BT_LITTLEFS_STORAGE_END_ADDRESS  (0x60800000)
 EDGEFAST_BT_LITTLEFS_STORAGE_SECTOR_SIZE    (0x1000)
```

```
#define m_flash_config_start            0x70000000
#define m_flash_config_size             0x00001000

#define m_ivt_start                     0x70001000
#define m_ivt_size                      0x00001000

#define m_interrupts_start              0x70002000
#define m_interrupts_size               0x00000400

#define m_text_start                    0x70002400
#define _ROM_END_                       0x7017FFFF
 EDGEFAST_BT_LITTLEFS_STORAGE_START_ADDRESS  (_ROM_END_ + 1)
 EDGEFAST_BT_LITTLEFS_STORAGE_END_ADDRESS  (0x70400000)
 EDGEFAST_BT_LITTLEFS_STORAGE_SECTOR_SIZE    (0x1000)
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

Migrate examples from RT1060EVKC to RT1064 This topic describes the Common steps and the steps to migrate with the IAR, Arm GCC, and MDK toolchains.

Common steps

1. Download SDK_2.14.0_EVKC-MIMXRT1060 and SDK_2.14.0_EVK-MIMXRT1064.
2. Copy the following folders from the RT1060EVKC package to the RT1064 package: <install_dir>/middleware/edgefast_bluetooth/ <install_dir>/middleware/wireless/ethermind.
3. Create a new folder named edgefast_bluetooth_examples/ under <rt1064_install_dir>/boards/evkmimxrt1064/.
4. Copy the entire folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/.
5. Copy clock_config.[c/h] and board.c from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/ to <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/ to replace the previous files.

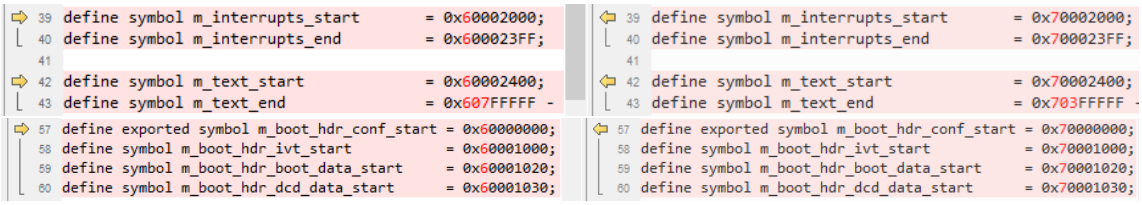
Parent topic:*Migrate examples from RT1060EVKC to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes in the listed order.

File name	Previous item	New item
peripheral_ht.ewp	1062	1064
mflash/evkcmimxrt1060	mflash/mimxrt1064	
evkcmimxrt1060	evkmimxrt1064	
6B	6A	

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.



Parent topic:*Migrate examples from RT1060EVKC to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.

2. Copy folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/template/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/ and rename the files.

Path	Previous name	New name
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/	middleware_edgefast_bluetooth_mcux_linker_template_evkcmimxrt1060.cmake	middleware_edgefast_bluetooth_mcux_linker_template_evkmimxrt1064.cmake
	middleware_edgefast_bluetooth_sdio_template_evkcmimxrt1060.cmake	middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064.cmake

3. Add the following content to<rt1064_install_dir>/devices/MIMXRT1064/all_lib_device.cmake at appropriate location.

```
...
  ${CMAKE_CURRENT_LIST_DIR}/../../boards
  ${CMAKE_CURRENT_LIST_DIR}/../../boards/evkmimxrt1064/edgefast_bluetooth_examples/
↪template
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/edgefast_bluetooth
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/wireless/ethermind
...
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci_uart)
include_if_use(middleware_edgefast_bluetooth_config_ethermind)
include_if_use(middleware_edgefast_bluetooth_config_template)
include_if_use(middleware_edgefast_bluetooth_extension_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_k32w061_controller)
include_if_use(middleware_edgefast_bluetooth_mcux_linker_template_ evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_pal)
include_if_use(middleware_edgefast_bluetooth_pal_db_gen_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_host_msd_fatfs_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_platform_ethermind)
include_if_use(middleware_edgefast_bluetooth_porting)
include_if_use(middleware_edgefast_bluetooth_porting_atomic)
include_if_use(middleware_edgefast_bluetooth_porting_list)
include_if_use(middleware_edgefast_bluetooth_porting_net)
include_if_use(middleware_edgefast_bluetooth_porting_toolchain)
include_if_use(middleware_edgefast_bluetooth_porting_work_queue)
include_if_use(middleware_edgefast_bluetooth_profile_bas)
include_if_use(middleware_edgefast_bluetooth_profile_dis)
include_if_use(middleware_edgefast_bluetooth_profile_fmp)
include_if_use(middleware_edgefast_bluetooth_profile_hps)
include_if_use(middleware_edgefast_bluetooth_profile_hrs)
include_if_use(middleware_edgefast_bluetooth_profile_hts)
include_if_use(middleware_edgefast_bluetooth_profile_ipsp)
include_if_use(middleware_edgefast_bluetooth_profile_pxr)
include_if_use(middleware_edgefast_bluetooth_profile_tip)
include_if_use(middleware_edgefast_bluetooth_profile_wu)
include_if_use(middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_shell)
include_if_use(middleware_edgefast_bluetooth_shell_ble)
include_if_use(middleware_edgefast_bluetooth_template)
include_if_use(middleware_edgefast_bluetooth_wifi_nxp_controller_base)...
```

4. Make the following changes in the listed order.

File name	Previous item	New item
config.cmake	MIMXRT1	MIMXRT1064xxxxA
mflash_evkcmimxrt1060	mflash_rt	
1062	1064	
evkcmimxrt1060	evk-mimxrt10	
flags.cmake	1062	1064
6B	6A	
CMakeLists.txt	1062	1064
<rt1064_install_dir>/middleware/edgefast_bluetooth/middleware_edgefast_bluetooth_template.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_common_ethermind_hci_uart.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_k32w061_controller.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_wifi_nxp_controller_base.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_mcux_cmake	1062	1064 mxrt1064.
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_sdio_t cmake	1062	1064 1.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.

```
m_text_start         = 0x60002400;
m_text_size          = 0x007FDC00 - LITTLEFS_REGION_SIZE;

HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x1000;
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x0400;
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400 : 0;

/* Specify the memory areas */
MEMORY
{
  m_flash_config       (RX)  : ORIGIN = 0x60000000, LENGTH = 0x00001000
  m_ivt                (RX)  : ORIGIN = 0x60001000, LENGTH = 0x00001000
  m_interrupts         (RX)  : ORIGIN = 0x60002000, LENGTH = 0x00000400
```

```
m_text_start         = 0x70002400;
m_text_size          = 0x003FDC00 - LITTLEFS_REGION_SIZE;

HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x1000;
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x0400;
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400 : 0;

/* Specify the memory areas */
MEMORY
{
  m_flash_config       (RX)  : ORIGIN = 0x70000000, LENGTH = 0x00001000
  m_ivt                (RX)  : ORIGIN = 0x70001000, LENGTH = 0x00001000
  m_interrupts         (RX)  : ORIGIN = 0x70002000, LENGTH = 0x00000400
```

Parent topic:*Migrate examples from RT1060EVKC to RT1064*

MDK

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.
2. Make the following changes in the listed order.

File name	Previous item	New item
peripheral_ht.uvprojx	1062	1064
	mflash/evkcmimxrt1060	mflash/mimxrt1064
	evkcmimxrt1060	evkcmimxrt1064
	6B	6A

3. Copy evkmimxrt1064_flexspi_nor.ini from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/mdk/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.
4. Rename MIMXRT1062xxxxx_flexspi_nor as MIMXRT1064xxxxx_flexspi_nor and make the following changes.

```
43 #define m_flash_config_start          0x60000000
44 #define m_flash_config_size           0x00001000
45
46 #define m_ivt_start                   0x60001000
47 #define m_ivt_size                    0x00001000
48
49 #define m_interrupts_start            0x60002000
50 #define m_interrupts_size             0x00000400
51
52 #define m_text_start                  0x60002400
53 #define m_text_size                   0x007FDC00 - LITTLEFS
```

```
43 #define m_flash_config_start          0x70000000
44 #define m_flash_config_size           0x00001000
45
46 #define m_ivt_start                   0x70001000
47 #define m_ivt_size                    0x00001000
48
49 #define m_interrupts_start            0x70002000
50 #define m_interrupts_size             0x00000400
51
52 #define m_text_start                  0x70002400
53 #define m_text_size                   0x003FDC00 - LITTLEFS
```

Parent topic:*Migrate examples from RT1060EVKC to RT1064*

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Enabling Additional Edgefast BT PAL Examples on M4 core for RT1170

Introduction RT1170 works with two cores: M7 and M4, on which both all EdgeFast examples can run. However, all the EdgeFast examples in the release package are enabled on M7. Only the A2DP source example is enabled on M4.

EdgeFast projects for both the cores share the demo source files but with different project settings. Therefore, the examples can be migrated.

This document describes the steps to migrate EdgeFast examples from M7 to M4 with different toolchains. There are four main steps required. Additionally, you can also delete the function.

1. Create an M4 project
2. Rearrange source files
3. Rearrange project files

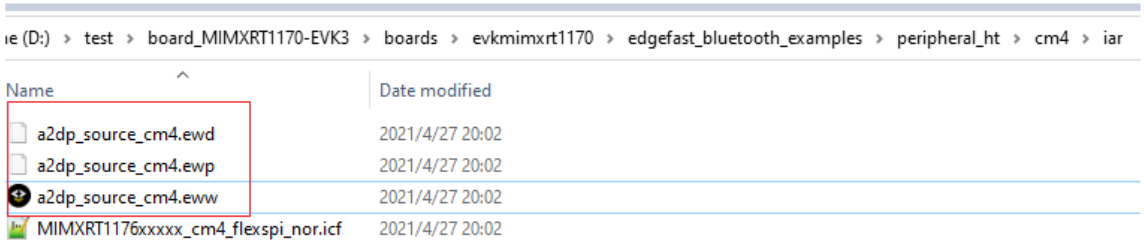
4. Adjust project settings
5. Delete function

In this document, the peripheral_ht example is used to demonstrate how to enable EdgeFast examples on M4 core with IAR and ARMGCC.

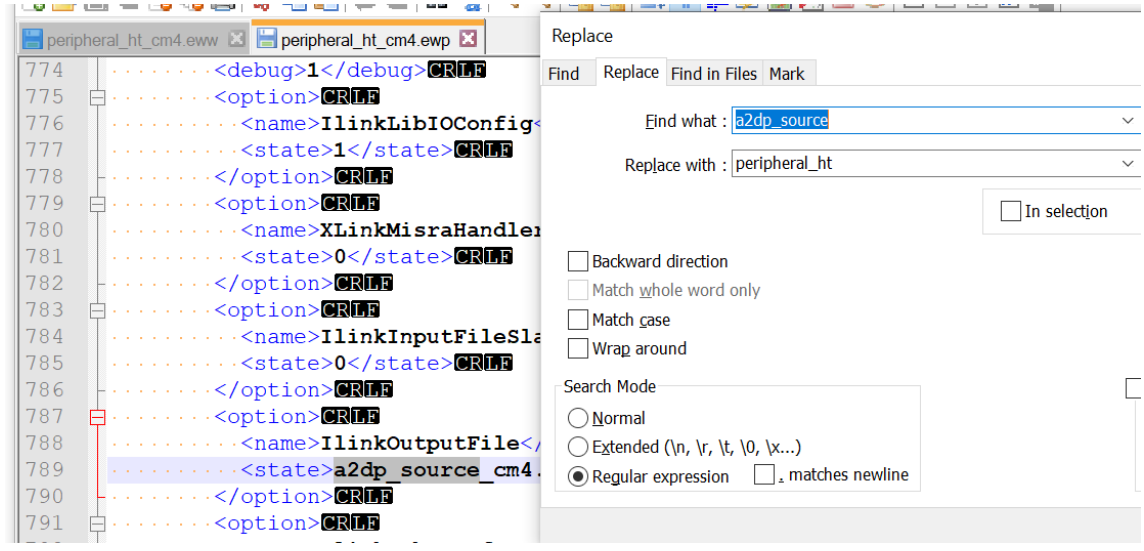
IAR This section describes the steps to create an M4 project with IAR, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project To create an M4 project, perform the following steps:

1. Copy the folder *cm4* in the directory <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*a2dp_* into the folder in which the example should be enabled. In this case, copy the folder *cm4* into the directory <*install_dir*>|*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*.
2. Open the folder *iar* in the directory <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*periph*
3. Rename the files. Change the file name name *a2dp_source_cm4* to *peripheral_ht_cm4* in all the respective files.



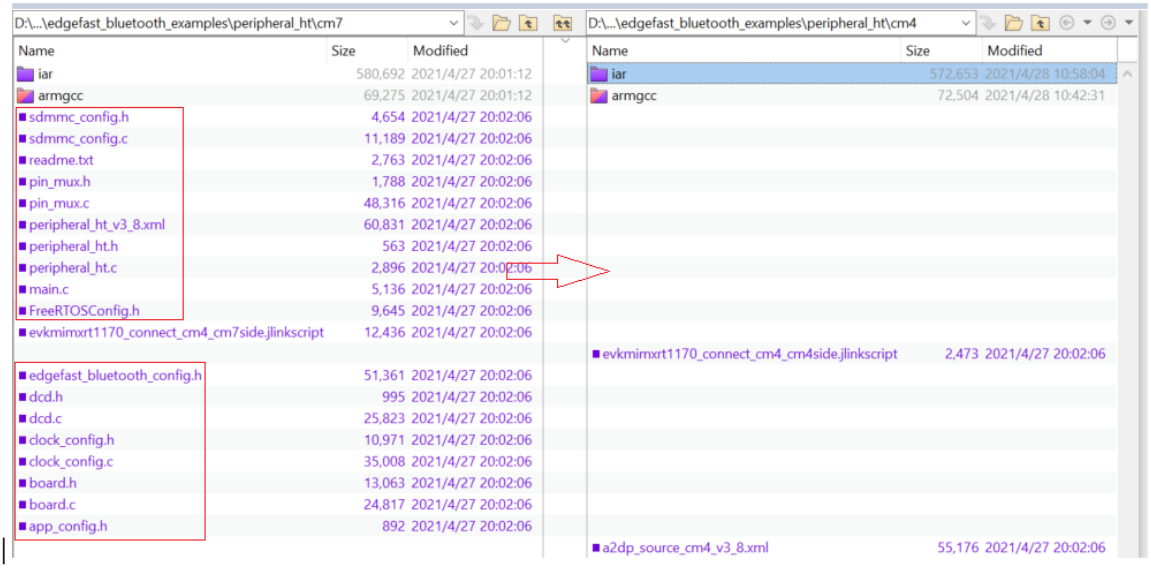
4. Open the files *peripheral_ht_cm4.eww* and *peripheral_ht_cm4.ewp* with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio Code.
5. Search and replace all *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.



Parent topic:*IAR*

Rearrange source files To rearrange source files, perform the following steps:

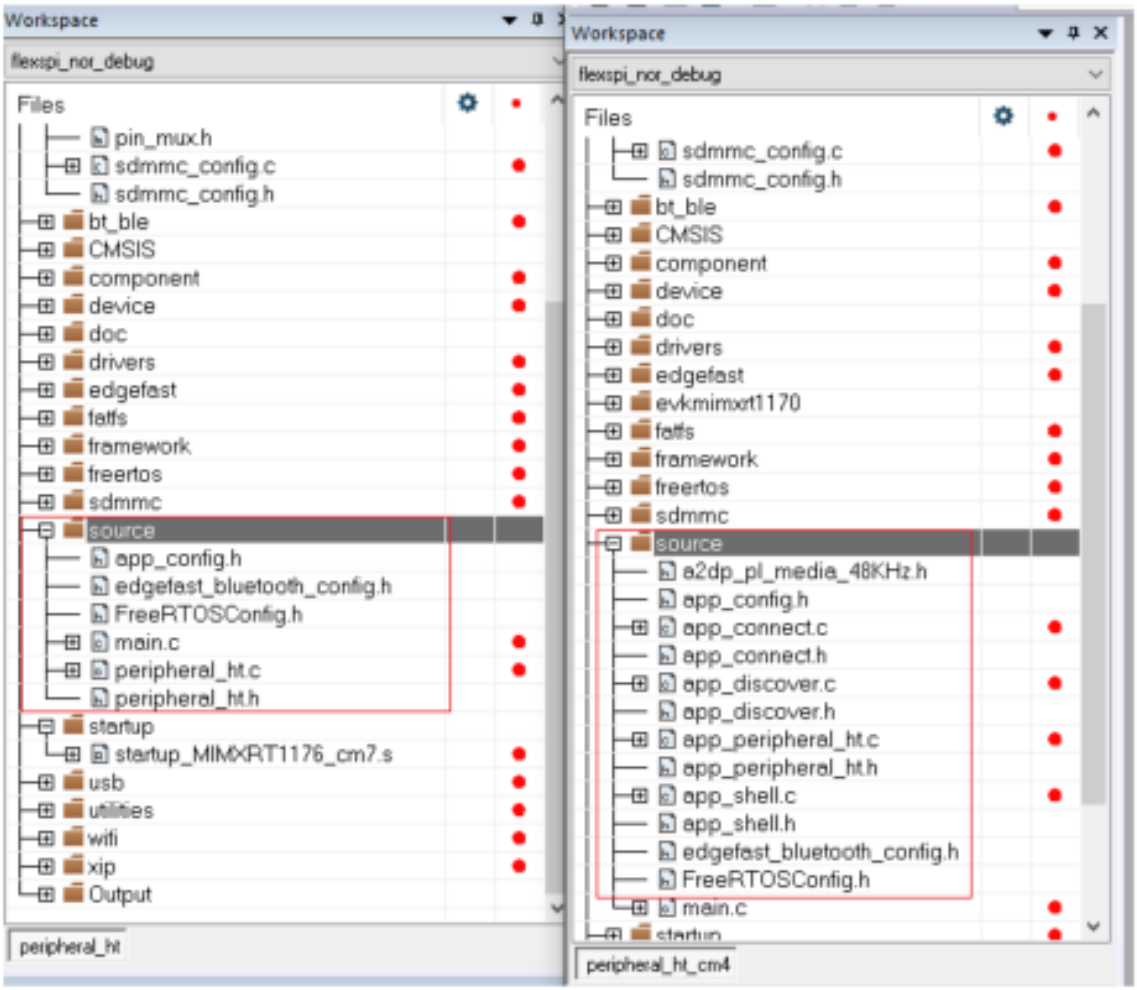
1. Open the folder *cm4* in the directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peri* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h from the folder *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm7*| to the folder <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*.



Parent topic:*IAR*

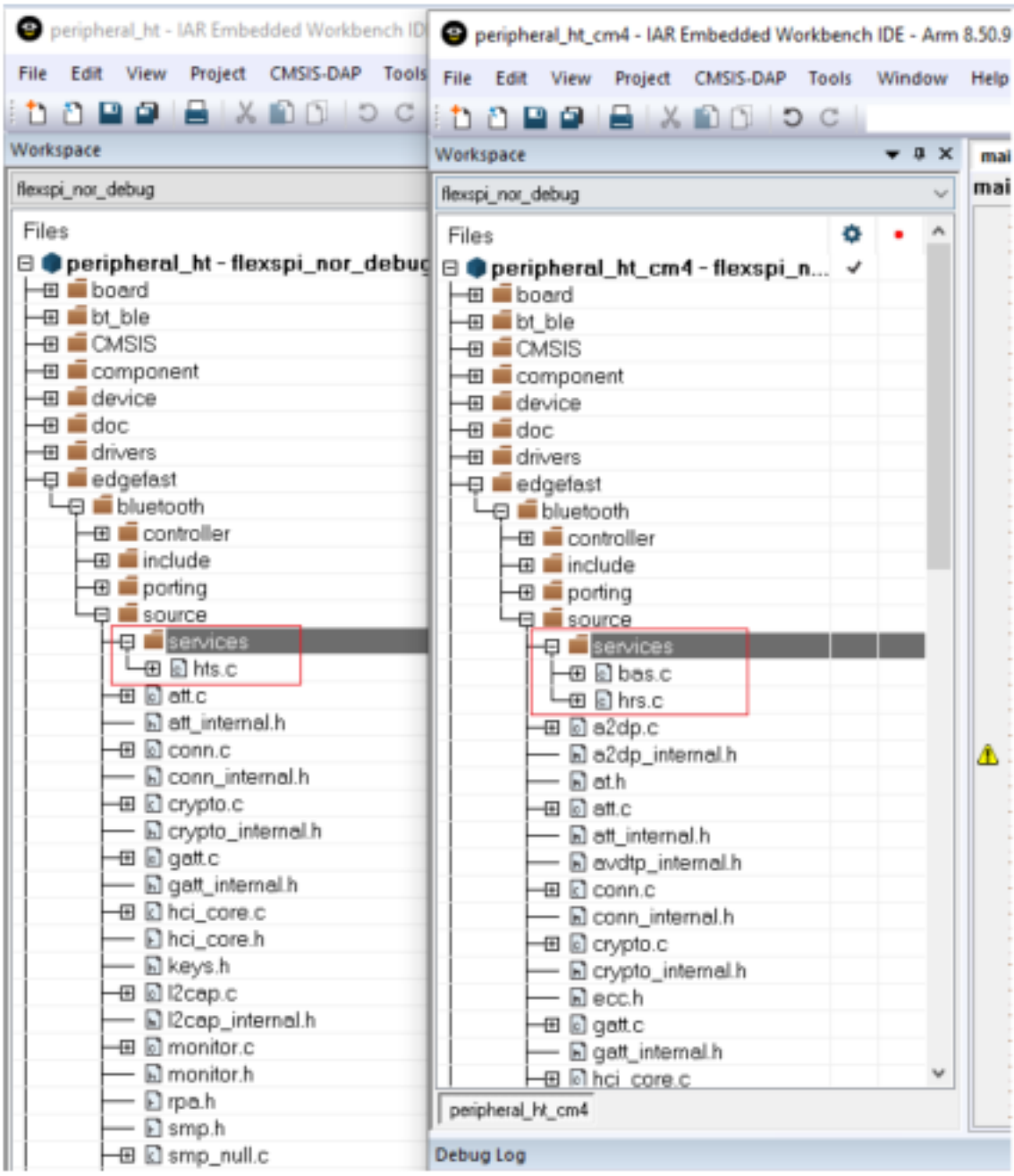
Rearrange project files To rearrange project files, perform the following steps:

1. Open the *peripheral_ht _cm7*and *peripheral_ht _cm4* IAR projects in the directories <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht* |*cm7*|*iar* and <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht* |*cm4*|*iar*.
    1. Compare the whole project directory, find file groups that the cm7 project has but are missing in the cm4 project. Add the missing file groups from the cm7 project into the cm4 project.
    2. Compare the difference between the two groups with the same name. Remove files that do not exist in the cm7 project but exist in the cm4 project. Find files that are available in the cm7 project but are missing in the cm4 project. Add the missing files from the cm7 project into the cm4 project.
2. For example, in the following figure, the files in the source group in the cm4 project must be removed, and the files in the path: <*install_dir*>|*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht* with the same name as the files in the cm7 project must be added into the *source* group.



3. Compare the *services* group.

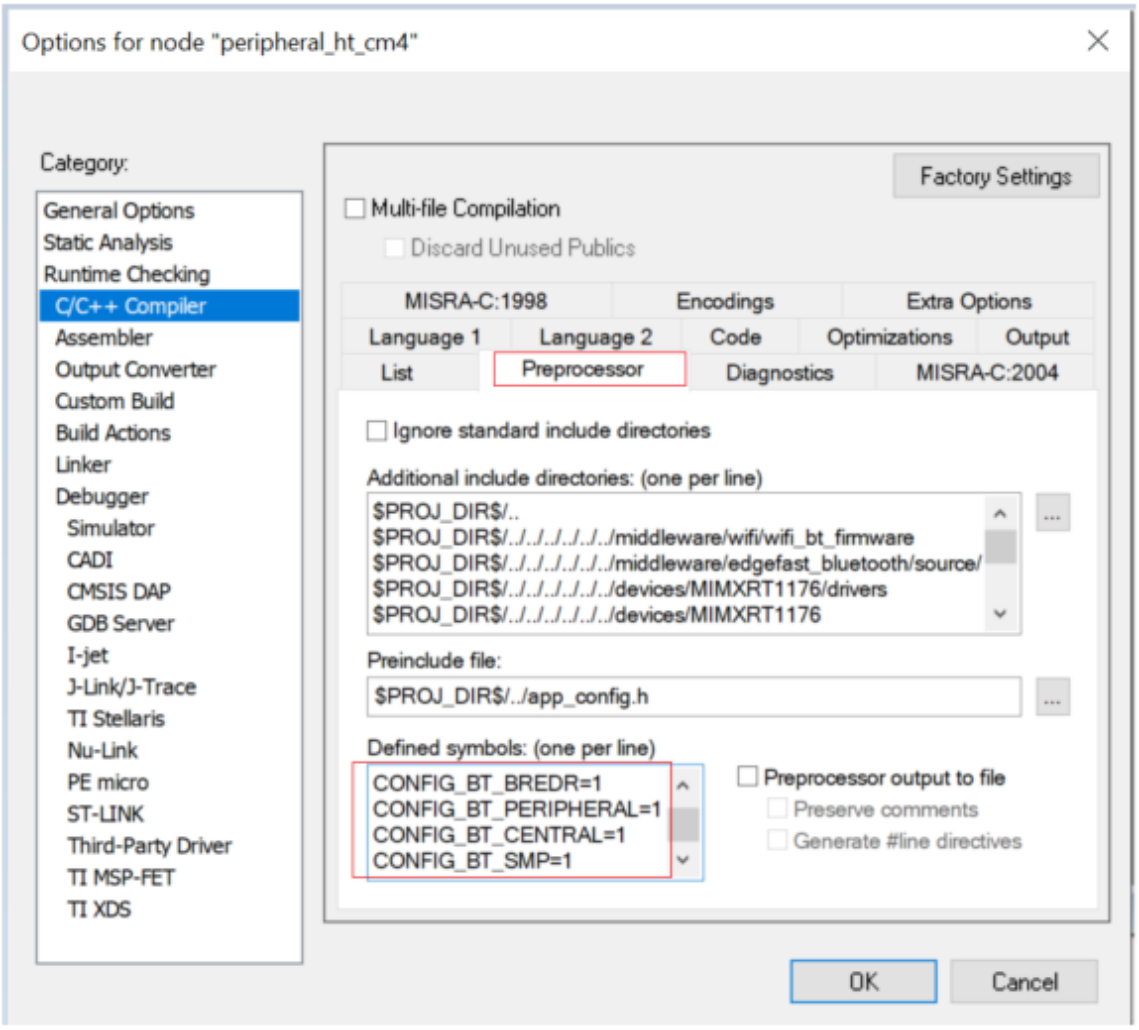
   The peripheral hts profile is in the *services* folder. Add the hts.c file to the *services* group of the *cm4* folder.



Parent topic:*IAR*

Adjust project settings To adjust the project settings, perform the following steps:

1. Compare the macro in the project settings: **Option** > **C/C++ compiler** > **Preprocessor**.
2. Find the macros that do not exist in the **cm4** project but are available in the **cm7** project. Delete these macro. The rule is that **m7** macro setting should be same with **m4**.



|

|

The macros are in the**peripheral_ht_cm4.ewp** file.

|

|

Parent topic:*IAR*

Delete function As a final step, remove the function *"SCB_DisableDCache()*; in main.c.

On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:*IAR*

Arm GCC This section describes the steps to create an M4 project with Arm GCC, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project To create an M4 project, perform the following steps:

1. Copy the folder <*install_dir*> *boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source\cm4* into another folder in which the example should be enabled. In this case, copy the folder <*install_dir*>

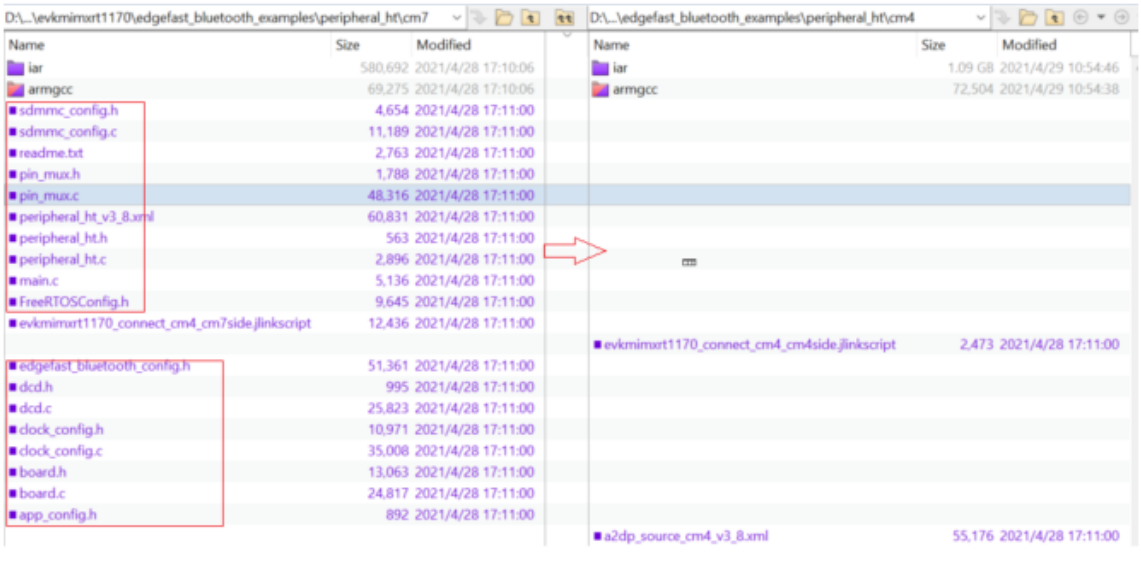
boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source \cm4 into*<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.

2. Open the file *CMakeLists.txt* located in the path: *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\ armgcc*.
3. Search and replace all *a2dp_source_cm4* with *peripheral_ht_cm4*, and then save the files.

Parent topic:*Arm GCC*

Rearrange source files To rearrange source files, perform the following steps:

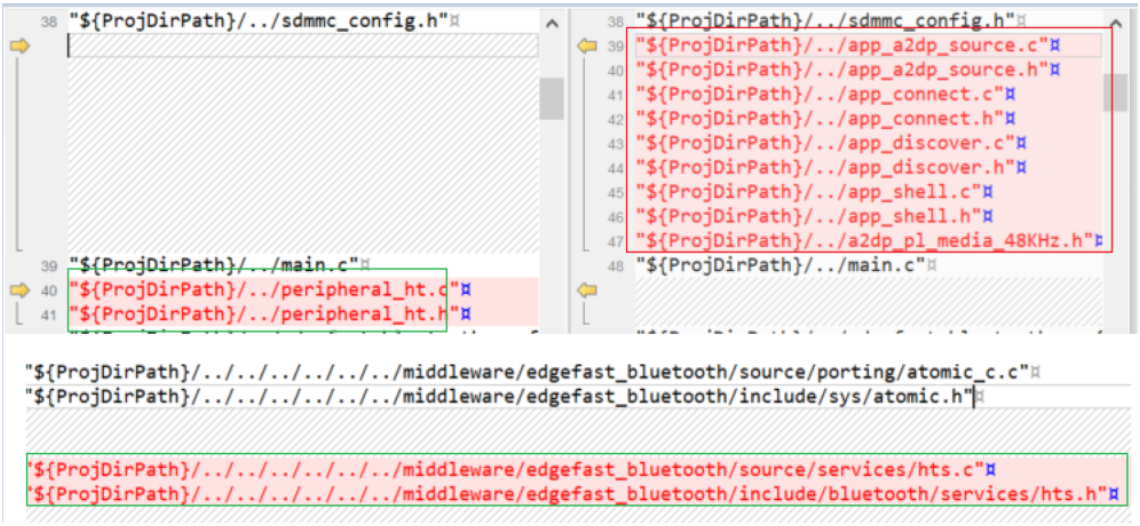
1. Open the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h in the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7* to the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.



Parent topic:*Arm GCC*

Rearrange project files To rearrange project files, perform the following steps:

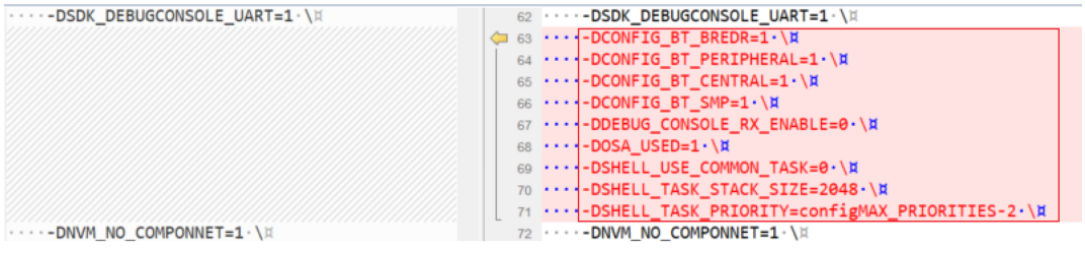
1. Open the *CMakeLists.txt* of the two examples respectively. The two files are in the *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7\armgcc* and *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\armgcc* folders respectively.
2. Search the section *add_executable*. Compare the difference between the two sections. Remove files that do not exist in the *cm7* project but are available in the *cm4* project. Add the files that exist in the *cm7* project but are not available in the *cm4* project into the *cm4* project. For example, in the following figure, the files in the red box should be removed and the files in the green box must be added into the *cm4* project.



Parent topic:*Arm GCC*

Adjust project setting To adjust the project settings, perform the following steps:

1. Open the *flags.cmake* of the two examples respectively. The two files are in the <*install_dir*>*boards**evkmimxrt1170**edgefast_bluetooth_examples**peripheral_ht**cm7**armgcc* and <*install_dir*>*boards**evkmimxrt1170**edgefast_bluetooth_examples**peripheral_ht**cm4**armgcc* folders respectively.
2. Search the ***CMAKE_C_FLAGS_DEBUG*** section.
    1. Compare the macro between the two sections.
    2. Add the macros that do not exist in the **cm4** project but are available in the **cm7** project into the cm4 project. The rule is that macro setting should be same.
    3. Delete the macros highlighted in the red rectangle.



Parent topic:*Arm GCC*

Delete function As a final step, remove the function *"SCB_DisableDCache()* in main.c.

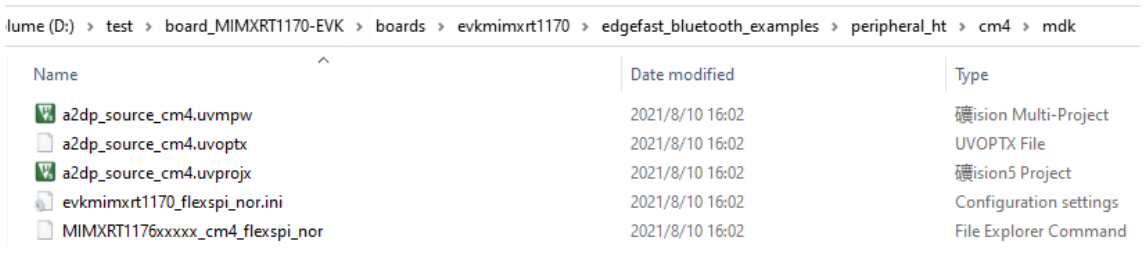
On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:*Arm GCC*

MDK This section describes the steps to create an M4 project with MDK, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project

1. Copy folder *cm4* from <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*a2dp_source*|*cm4* into the folder in where the example must be enabled. In this case, copy folder *cm4* into directory <*install_dir*>|*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*.
2. Open folder mdk from <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*

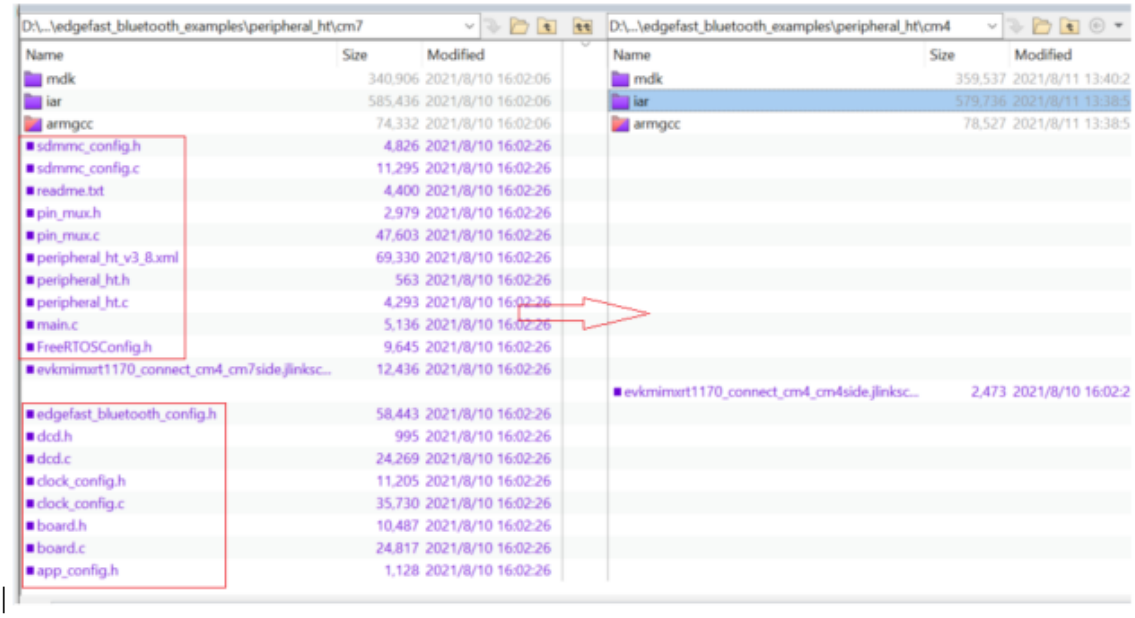


3. Change the filename *a2dp_source_cm4* to *peripheral_ht_cm4* respectively.
4. Open the files *peripheral_ht_cm4.**uvmpw* and *peripheral_ht_cm4.* *uvoptx*, *peripheral_ht_cm4.uvprojx*with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio code.
5. Search and replace *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.

Parent topic:*MDK*

Rearrange source files

1. Open folder *cm4* in *<install_dir>**boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*, and delete all files with the .c and .h file name extension.
2. Copy files with the .c and .h filename extension in folder *cm7* with directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm7*to folder *cm4* with directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*.

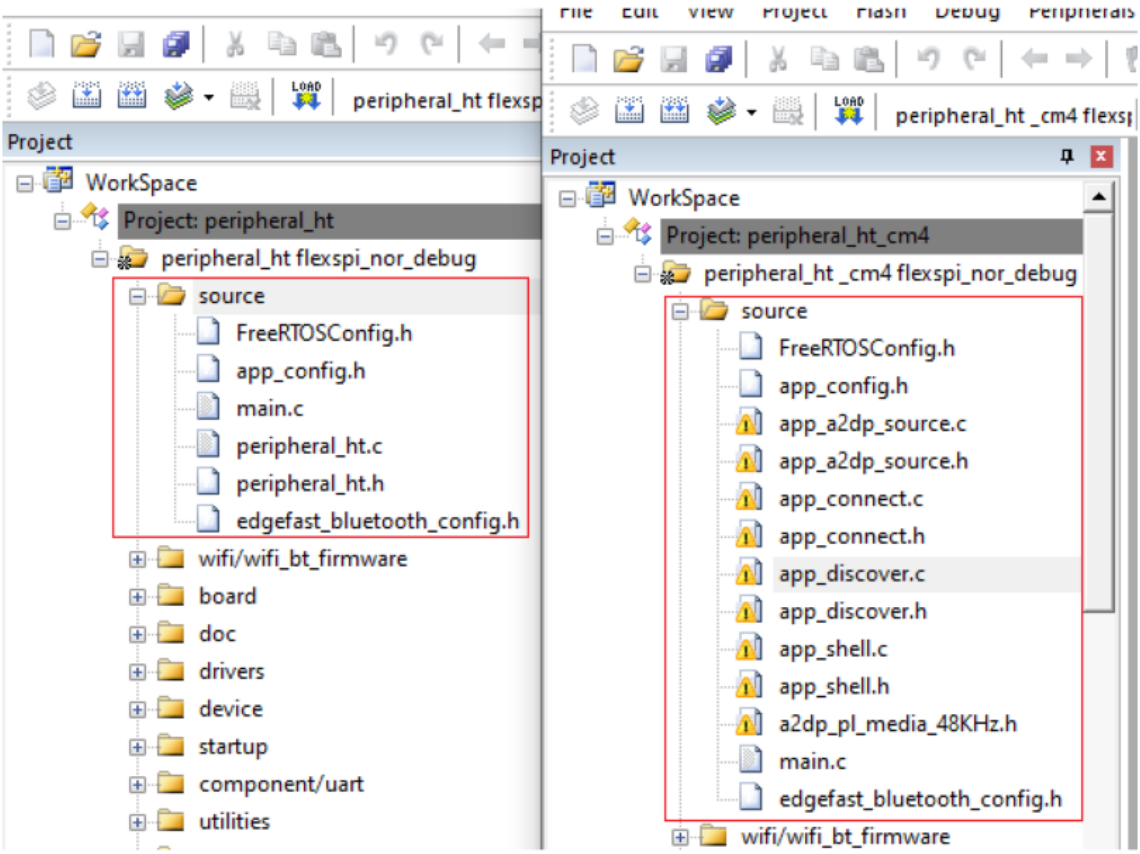


.

Parent topic:*MDK*

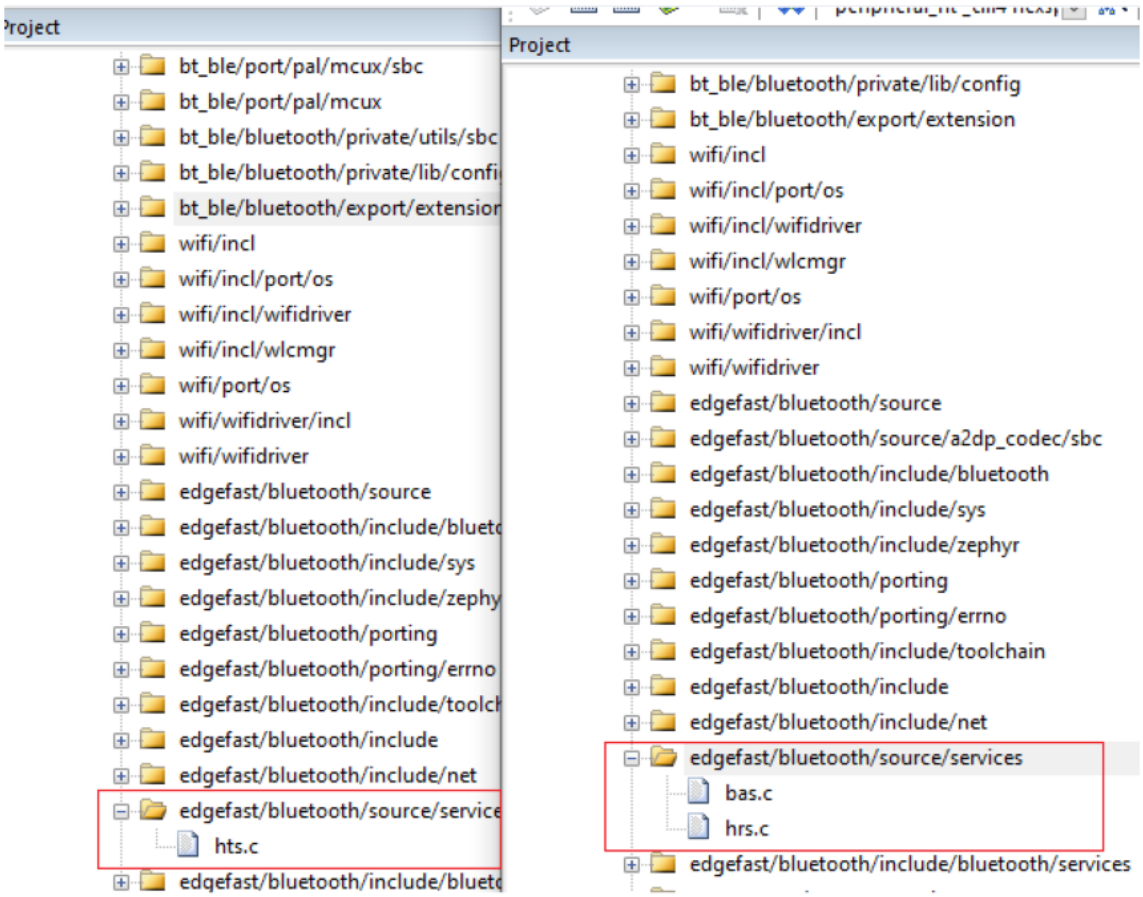
Rearrange project files

1. Open the *peripheral_ht _cm7* and *peripheral_ht _cm4* IAR projects. The two workspaces are located in *<install_dir>**boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm7*|*mdk* and *<install_dir>**boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*|*mdk* respectively.
   - Compare the whole project directory, find file groups that the cm7 project has but the cm4 project not and then add these groups into the cm4 project.
   - Compare the difference between the two groups with the same name, remove files that do not exist in the cm7 project but exist in the cm4 project; find files that the cm7 project has but the cm4 project not and then add these files into the cm4 project.
2. For the *source* group, in this case, the files in the source group in the cm4 project must be removed, and the files in the path *<install_dir>*|*boards*|*evkmimxrt1170*|*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4* with the same name as the files in the cm7 project must be added into the *source* group.



3. Compare the **service: group**.

   Peripheral hts profile is located in "**service**" folder. Add the hts.c file to the services group of the cm4 folder.

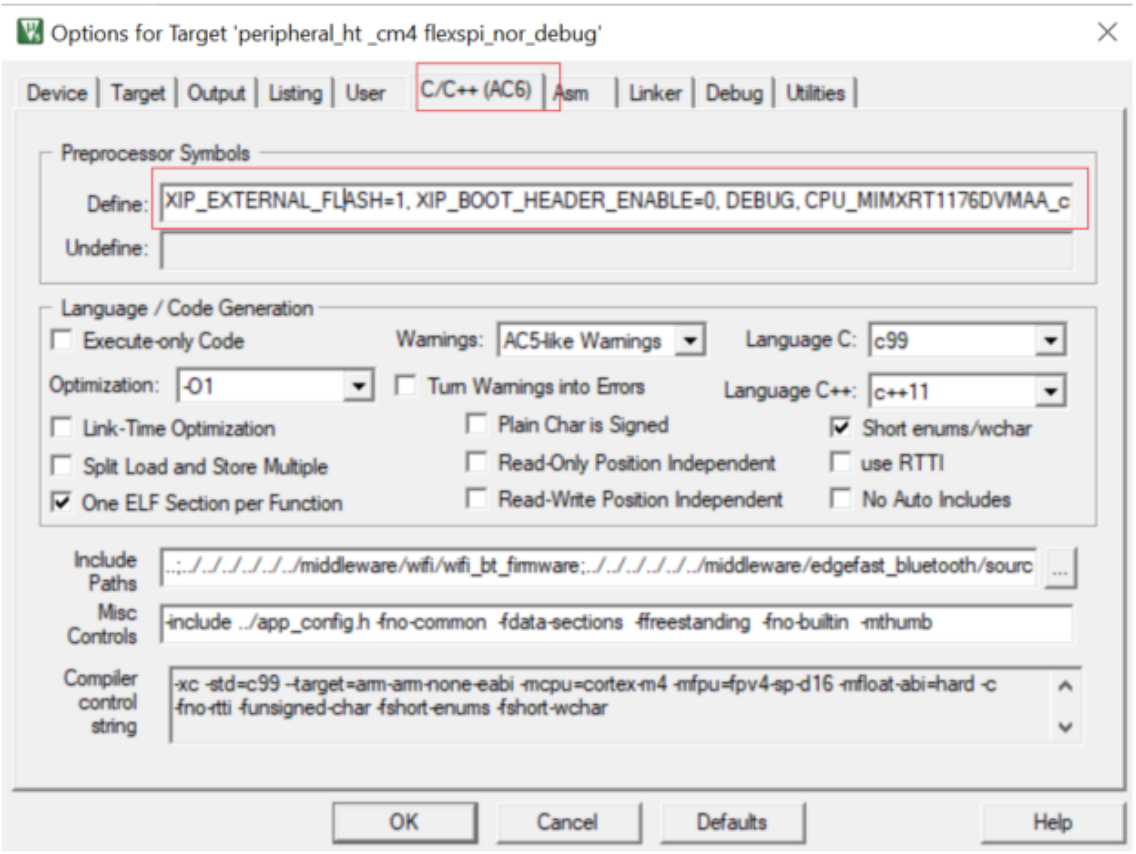


.

Parent topic:*MDK*

Adjust project settings

1. Compare the macro in the project settings: *preprocessor symbols*.
2. Compare the macro that does exist in the cm4 project but exists in the cm7 project.
3. Delete the following macro. The rule is that m7 macro setting should be same as m4 .
   The macro could also be found in be eripheral_ht_cm4.uvprojx.



|

|

|

Parent topic:*MDK*

Delete function Remove function *SCB_DisableDCache();* in *main.c*.

On successful completion of the above steps, the M7 project is changed to the M4 project. You can now download and debug the M4 example project.

Parent topic:*MDK*

Note The above steps are based on the **a2dp_source** example and help enable the **peripheral_ht** example on the **m4** core. You can use the same steps for other examples and migrate them from an **m7** project to an **m4** project.

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